Georgia Tech Sponsored Research

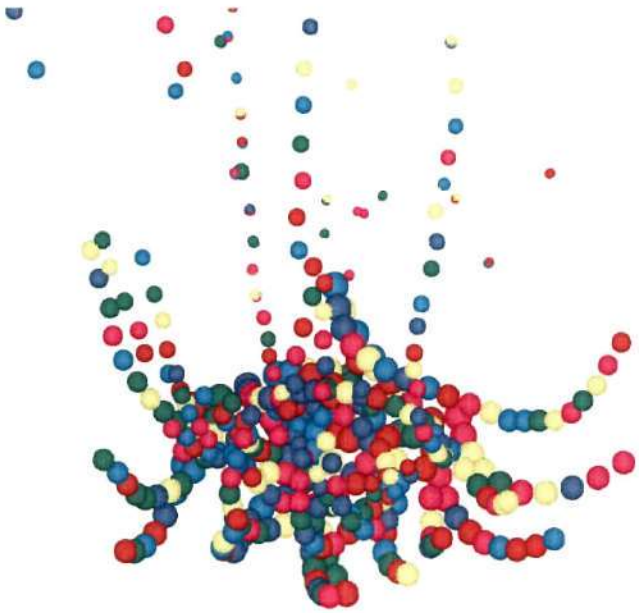
64020307

Project	E-20-E10	S5985	25
Project director	Sotiropoulos	Fotis	
Research unit	CEE		
Title	Coupled Two-Phase Flow Numerical Model for Predicting DO Transfer in Autoventing Hydroturbines		
Project date	8/31/2000		

COUPLED TWO-PHASE FLOW NUMERICAL MODEL FOR PREDICTING DO TRANSFER IN AUTOVENTING HYDROTURBINES

by
Y. Ventikos and F. Sotiropoulos

Final report for project E-20-E10



Sponsored by:
Voith Hydro, Inc.

Environmental Hydraulics and Water Resources Group
School of Civil and Environmental Engineering
Georgia Institute of Technology
Atlanta GA 30332-0355

December 1999

COUPLED TWO-PHASE FLOW NUMERICAL MODEL FOR PREDICTING DO TRANSFER IN AUTOVENTING HYDROTURBINES

by
Y. Ventikos and F. Sotiropoulos

Final report for project E-20-E10

Sponsored by:
Voith Hydro, Inc.

Environmental Hydraulics and Water Resources Group
School of Civil and Environmental Engineering
Georgia Institute of Technology
Atlanta GA 30332-0355

December 1999

Table of contents

1. Executive Summary

This report describes a numerical procedure for the determination of the flow characteristics and Dissolved Oxygen transfer in Autoventing HydroTurbine draft tubes. The method is based on the combination of a Reynolds Averaged Navier-Stokes solver along with a three-dimensional bubble tracking Lagrangian model, that accounts for oxygen transfer, bubble breakup and coalescence, and the exchange of information between the two via the computation of appropriate momentum source terms.

The purpose of this algorithm is to provide a tool for the fast and easy estimation of losses in autoventing draft tube, thus the resulting computer environment had to be very efficient and facilitate repetitive use. This need has guided all the modeling choices that are described in subsequent sections of this report. It should be emphasized, however, that the model has been constructed in a modular form so that its various components can be readily enhanced as additional data or more refined models become available.

2. Introduction

This report describes the development and application of a numerical method for the prediction of Dissolved Oxygen (DO) transfer and flow characteristics in autoventing hydroturbine (AVT) draft tubes. The technique presented herewith falls under a general four-step approach that is employed in order to account for the water-air coupling and interaction: for the first step, the Reynolds Averaged Navier-Stokes (RANS) equations for the single phase (water) flow through the draft tube are solved using any suitable fluid solver. The second step involves a Lagrangian methodology for tracking an arbitrary number of air-bubbles through the steady flowfield obtained during the previous stage. The third stage of the procedure involves the computation of suitable momentum source terms that correspond to the existence of the air bubbles in the draft tube. These source terms are correlated with the fully developed distribution of the bubbles as obtained from the previous step and account for the drag that such spherical bubbles apply to the surrounding fluid. During the fourth and final step of the procedure, the single phase flow field is solved again using the RANS solver, only this time the existence of the aforementioned source terms is taken into account, thus accounting for the influence of the bubble distribution on the water flow. It is assumed, throughout this report that the first and fourth step of the procedure just described are fully outlined elsewhere and require no additional documentation. In effect, the focus here is on the second and third step, i.e. on the bubble tracking algorithm an on the estimation of the momentum source terms distribution.

The Lagrangian bubble tracking algorithm accounts for most of the physical processes that govern the motion and oxygen exchange of the air bubbles. These bubbles are allowed to coalesce, break-up into smaller bubbles and exchange DO with the water, under the assumption that their motion does not alter the local flow characteristics, during this particular step of the procedure. This influence, or coupling, is the reason the next two steps of the procedure are performed, as described in the sequel. A second assumption which is implied from the previous one is that the air transferred from the bubbles to the water does not alter the DO concentration of the water. In other words, the ability of the water to absorb DO at a given instant in time is not altered by the amount of DO that has been already dissolved at earlier times. The validity of this assumption and its impact on the computed DO transfer rates requires further investigation, particularly when the air-water mixture approaches saturation conditions. One may speculate, however, that since the residence time of the bubbles inside the draft tube is small such an assumption may not significantly affect the computed results.

The above simplifying assumptions are crucial for the computational efficiency of the present model. A more exact treatment would necessitate: i) the use of the so-called simultaneous two-way coupling approach, in which the air and water phases are coupled together through source terms in their respective transport equations and solved for at the same time; and ii) the solution of a transport equation for the DO concentration of the water in order to account for concentration history effects. Such a level of sophistication would obviously increase significantly the required computational resources, particularly since our objective is to develop a practical engineering tool that can be used to optimize the design of AVT draft tubes. Typical

AVT draft tubes are geometrically very complex, include multiple downstream piers, and feature a number of air-injection outlets. Furthermore, obtaining statistically meaningful results for such a complex geometrical configuration requires carrying out simulations with at least few thousands of air-bubbles. Thus, the main challenge that we had to address in this work was to strike a fine balance between the accuracy of the computed results and the computational efficiency and expedience of the overall numerical model. This need has guided all the modeling choices that are described in subsequent sections of this report. The present model, although simpler than others reported in the literature (see Domgin et al. (1997) for a recent review), at least as far as the bubble tracking algorithm is concerned, is the first attempt to apply such methods to complex three-dimensional flows. Previous studies have primarily focused on simple straight pipe geometries. It should be emphasized, however, that the model has been constructed in modular form so that its various modules can be readily enhanced as additional data or more refined models of various physical processes become available.

In what follows, we start by describing the bubble tracking and DO transfer models and present and discuss representative results from the application of the model to a typical AVT draft tube. At the end of this report, we provide a detailed user's manual and a copy of the entire computer code developed to implemented the present model.

3. The bubble tracking algorithm

The numerical method requires as input a complete three-dimensional solution for the single-phase draft tube flowfield--in terms of pressure, mean velocity components, and turbulence statistics--at a given powerplant operating point. The precomputed flow comprises the Eulerian component of the present model and is obtained by employing an existing RANS draft tube flow solver (Ventikos et al., 1996, or TASCFLOW manual, 1997). Discrete air bubbles are subsequently introduced in this virtual flow environment at user specified locations. The bubbles are released in a time accurate manner, so that the total amount of air they carry into the flow per time step corresponds to the desired air discharge. The trajectory of each bubble is computed using a Lagrangian tracking algorithm. The motion of each bubble is described in terms of a sequence of translations along the three Cartesian axes, and, thus, a total of three ordinary differential equations (for the Cartesian components of the linear acceleration vector) are necessary for describing the entire spectrum of possible motions. The source terms in these equations represent the various forces exerted by the flow on the individual bubble. At every point along the computed trajectory the amount of DO transferred from the bubble to the surrounding water is monitored by solving a mass transfer equation. The application of this algorithm continues until one of the following events occurs:

- the bubble exits the computational domain, i.e. exits the draft tube;
- the bubble is depleted of all the air, and thus vanishes;
- the bubble approaches another bubble closer that a prescribed threshold and merges with it. From this time step on, the new bubble is tracked, having inherited properties from both the merged bubbles;
- the bubble encounters local conditions that lead to its splitting or fragmentation. Each resulting bubble is tracked individually from that point on;
- the bubble "sticks" to a solid wall leading to the formation of an air pocket.

It is evident from the above brief summary that the overall algorithm consists of several modules that need to be carefully formulated for accurate, physically meaningful predictions. These include the: i) selection of a statistically average bubble shape; ii) modeling of the bubble-injection process; iii) physics of the DO transfer and bubble dynamics; and iv) formulation and accurate and efficient numerical solution of the equations of motion. The modeling strategies adopted for each of these modules are described in detail in the subsequent sections.

3.1 Selection of a statistically average bubble shape

Numerous experimental observations (Shinnar, 1961, Maxworthy, 1991, Jun and Jain 1993) have shown that, depending on the local flow characteristics, the history of the bubble etc., air bubbles in water can have various regular and irregular shapes (see Fig. 1). Obviously it would be impractical to try to simulate the precise shape of each individual bubble in a model that must be applicable to very complex flows. An obvious first approximation would be to try to match the bubble shape with some kind of statistical average derived from experimental observations. Such a mean shape is believed to exist (Jun and Jain 1993) and is of the general

shape of an oblate spheroid, curve (d) in Fig. 1. Even this level of approximation, however, is not feasible, because:

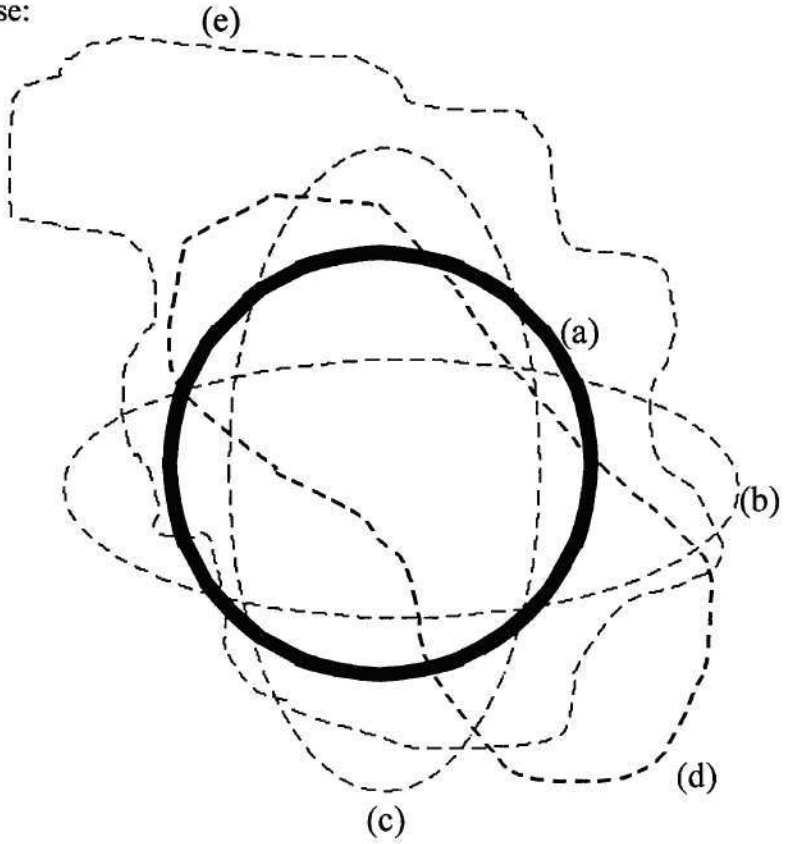


Figure 1. Bubble shapes

i) it would require accurate description of the bubble shape, which would drastically increase the required computational resources; and ii) there are no comprehensive estimates for the viscous drag force of such a body for all possible local flow conditions. These restrictions dictated the use of a spherical bubble (Shinnar, 1961), curve (a) in Fig. 1, as the core of this model.

3.2 Modeling of the bubble-injection process

Accurate modeling of the bubble injection process is of crucial importance for evaluating the performance of various aeration strategies. There are several parameters that must be either specified by the user or calculated in order to develop a meaningful bubble-injection model. These include: i) the location and geometrical characteristics of the air-injection orifices; ii) the frequency at which bubbles are injected into the flow; iii) the amount of total airflow; and iv) the initial bubble size.

The location and general geometric characteristics of the injection orifices are specified by the user. Since such orifices are in general arbitrarily shaped, we adopt herein a simple approach for approximating their geometrical shape. As shown in Fig. 2, we employ an ensemble of circular openings to approximate the exact shape of a given injection slot. In the present version of the model injection slots have been introduced at the deflector, the discharge edge of bucket, and the periphery of the draft tube inlet--obviously, the first two locations rotate

with the angular velocity of the runner. As already discussed, the model has been formulated such that other injection outlets can be introduced and tested in a rather straightforward manner.

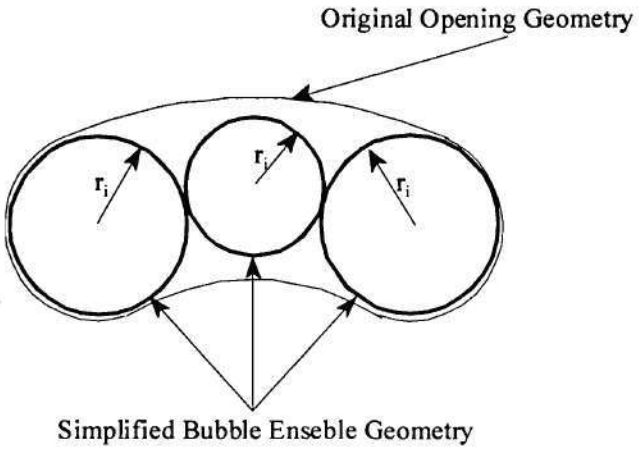


Figure 2. Typical simplified geometry of air injection openings

The frequency of bubble-injection, i.e. the number of time-steps between two successive injection events, as well as the air flowrate through each orifice are specified by the user. The actual number of bubbles that are introduced through an orifice when an injection event takes place is determined by the air flowrate and the size and properties of each bubble (see discussion in following paragraphs).

The issue of determining the exact size of the bubbles when they are injected into the flow field is very complicated and can be properly addressed only via experimental work. Existing experiments (Maxworthy 1991) provide some short of estimate of potential bubble sizes with respect to local flow conditions, but are rather case-specific and not straight-forward to apply. Thus, we decided to resolve this issue in a somewhat empirical manner. After experimenting unsuccessfully with various techniques (scaling with the air flowrate through each opening, the radius of the opening, characteristic times of the water flowrate etc.), we decided to simply treat the initial bubble size as an input, user-specified parameter that can be selected from experience and observations from model-scale laboratory experiments. Obviously, since the user has to also determine the distribution of the injection locations and the distribution of the total airflow among these locations, the initial bubble size should be selected so that the total number of bubbles introduced per time step corresponds to the desired total air flow rate.

It should be emphasized that due to the lack of a definitive approach for selecting the initial bubble size, the present version of the model can only be used to provide general qualitative trends. Obtaining accurate quantitative information about the actual air transfer taking place requires, among other things discussed subsequently, a physically-based approach for determining the initial bubble size.

3.3 Modeling of the various physical processes

3.3.1 Air exchange mechanism

As the bubble moves through the draft tube, it continuously transfers air to the surrounding water through its surface, which is the interface of the two phases. This exchange is governed by a physical law of the form:

$$\frac{dm}{dt} = k_L S(C_{sat} - C) \qquad (1)$$

That is, the rate of mass transfer through the interface is proportional to the surface area of the interface, S, and to the difference between the saturation air concentration of water, $C_{sat,}$, and the surrounding water air concentration C (Jun and Jain, 1993). A model for the mass transfer coefficient k_L, has been proposed by Jun and Jain (1993) as follows:

$$k_L = 8.33 \cdot 10^{-5} R^{0.363} \lambda^{-0.225} U \qquad (2)$$

where R is the flow Reynolds number, λ is the air-to-water flowrate ratio, and U is an average fluid velocity. Since the velocity varies greatly inside the draft tube (due to the diffuser effect of the geometry), U in eqn. (2) is set equal to the local relative velocity of the bubble, $U=U_{fluid}-V_{bubble}$. The air to water ratio is computed as the ratio of air flowrate Q_{air} over the total air and water flowrate ($Q_{water}+Q_{air}$).

The air concentration of the surrounding water, C, is assumed to be a user-specified constant that represents the air concentration of the water upstream of the draft tube. As already discussed, this treatment is only approximate since, in reality, C changes continuously with time via convection, molecular diffusion, turbulent transport, and transfer from passing bubbles. This assumption, however, should be reasonable for low air-fraction flows that evolve very rapidly, as is the case in typical AVT draft tube flows. Finally, the saturation concentration C_{sat} is also assumed to be constant and provided as a datum to the model. There is room for improvement here, however, since it is known (Baird and Rohatgi, 1989) that the saturation concentration is pressure sensitive. Such a sensitivity can be readily accounted for by incorporating in the model available in the literature tabulated data.

A significant amount of code infrastructure has been developed for using the results of the above local transfer model to estimate the total amount of air transfer from the bubbles to the water. More specifically, algorithms for calculating the total amount of air transferred to the water as well as the amount of air still trapped in bubbles exiting the draft tube have been incorporated and tested. Note that if the available computer resources do not permit a full simulation, i.e. releasing and tracking a total number of bubbles corresponding to the total air flowrate, the model can estimate the total DO transferred to the water using information from a partial simulation (i.e. a simulation using fewer bubbles than those needed for a full simulation). In such a case, the final dimensional amount of dissolved air as well as an estimate of the DO concentration at the exit of the draft tube (based on the water flow rate) can be calculated by

scaling the results of the partial simulation to the number of bubbles corresponding to the full airflow rate.

3.3.2 Bubble coalescence

Bubble coalescence occurs when the distance between two bubbles becomes sufficiently small so that the local flow field of each bubble affects the other one. As two bubbles are driven by the flow close to each other, the increased fluid velocity in the gap between them results to a local pressure drop (Bernoulli effect), thus, giving rise to a force that tends to bring the bubbles even closer.

This very complex body-fluid interaction mechanism is simulated in the model by implementing a very simple algorithm that checks the inter-bubble distance for all bubble pairs. Coalescence takes place when this distance becomes smaller than the sum of the two radii. The Bernoulli effect in this process is accounted for by introducing an effective bubble radius which is computed as the product of the actual radius times an empirical constant coefficient. This coefficient is greater than unity so that it increases the coalescence potential radius of each bubble.

An assumption implicit to the above model is that coalescence occurs only in a binary fashion. That is, following Shinnar (1961), at each time step only bubble pairs are checked for proximity. Once the two bubbles have joined together, apart from adjusting air content and radius, all of the other characteristics of the new bubble are inherited from one of the two parent bubbles in an ad hoc manner.

It must be noted here that the coalescence model is computationally very intense as it involves an exhaustive search for all possible bubble pairs during each time step. Incorporating this mechanism, however, is of crucial importance for realistic simulations particularly when there is significant residual swirl at the exit of the runner. This is because bubbles that are either released (deflector aeration) or transported by the flow near the core of the swirling flow, experience an imbalance between the centrifugal force an the radial pressure gradient and tend to move toward the vortex core and coalesce.

3.3.3 Bubble break-up

There are several bubble breakup models in the literature (Shinnar, 1960, Hughmark, 1971, Luo and Svendsen, 1996). These models range from relatively simple concepts, linking size with breakup, to very sophisticated treatments that rely on statistical considerations of eddy sizes and intensities. Since computational efficiency is of major interest herein, a compromise between level of sophistication (which in general is equivalent to accuracy) and performance had to be made. The breakup model finally employed is based on the concept of bubble critical diameter proposed by Hesketh et al. (1991a,b). According to this model, bubble break-up will occur when the bubble radius exceeds a threshold level, r_{crit}, given by the following equation:

$$r_{crit} = \frac{1}{2}\left(\frac{We_{crit}}{2}\right)^{0.6} \frac{\sigma^{0.6}}{\left(\rho^2_{water}\rho_{air}\right)^{0.2}} \bar{\varepsilon}^{-0.4} \qquad (3)$$

where σ is the surface tension of the air water interface (a constant in the model), We_{crit} is the critical Weber number set equal to 1.1 (Hesketh et al., 1991a), $\bar{\varepsilon}$ is the local energy dissipation rate, and ρ_{water} and ρ_{air} are the water and air densities, respectively.

Experimental observations (Hesketh et al. 1991b) show that when a bubble passes through a region where the local flow conditions are suitable for inducing breakup, the actual splitting does not occur instantaneously. Rather, it takes place after a small time delay which ranges from a fraction of a second to 10-11 seconds. This time delay parameter is a hard-coded constant in the present model. We should point out that our experience so far with the model has shown that this parameter, which to a large extent governs the rate at which bubbles break-up, is of great importance for determining the overall rate of DO transfer. This is because broken-up bubbles tend to exchange air with the water at a much faster rate. Thus fine-tuning the time delay constant should be among the first priorities for further enhancing the model. Such an undertaking, however, will require detailed experimental data which are not currently available.

It is implied again here that bubble breakup occurs in a binary fashion, i.e. each bubble marked for breakup, splits only to two new bubbles. The radii and air contents of the new bubbles are obtained by equi-distributing the air mass of the parent bubble. A statistically sound random distribution might make the model more realistic (Luo and Svendsen, 1996). All the other properties of the new bubbles are inherited from the parent bubble, except their spatial positions and position histories. These are determined by assigning the values of the original bubble to one of the two new bubbles and displacing the other one by a constant proportion of the radius of the original bubble, biased towards the center of the draft tube. This approach can be also improved by adopting a statistically rigorous distribution of the new bubbles. The new bubbles are re-initialized with respect to delay time for possible subsequent breakup, even if the local flow conditions dictate a second breakup to occur immediately.

3.4 The equations governing bubble motion

Since the bubble shape is assumed to be spherical and a sphere is invariant under rotation, only three differential equations, for the Cartesian components of the linear acceleration vector, are needed to fully describe the motion each bubble. Assuming steady flow, these equations are formulated as follows:

$$m_b \frac{du_{bi}}{dt} = F_D^i + F_P^i + F_{AM}^i + F_B^i + \cdots, \quad i = 1,2,3 \qquad (4)$$

where m_b is the mass of the bubble, d/dt is the Lagrangian derivative, u_{bi} are the Cartesian components of the bubble velocity vector, and the terms in the right hand side of eqn. (4) represent the various forces acting on the bubble at a given point along its trajectory. These include, forces due to: i) viscous drag, F_D^i; ii) ambient pressure gradient in the flow, F_P^i; iii) added mass effects, F_{AM}^i; and iv) buoyancy, F_B^i. The dots in eqn. (4) represent higher order forces that are typically difficult and time consuming to compute while their overall effect on the bubble trajectory is fairly small. For the sake of expedience and computational efficiency such forces are neglected herein. A complete review of the various forces acting on a spherical bubble can be found in the recent paper by Michailidis (1997).

Assuming that the various forces acting on the bubble are known, eqn. (4) can be integrated in time to obtain the bubble velocity at the new time steps. The new position of the bubble can subsequently determined by integrating in time the following equations for the Cartesian components, x_{bi}, of the bubble position vector:

$$\frac{dx_{bi}}{dt} = u_{bi}, \; i = 1,2,3 \tag{5}$$

The details of the numerical technique employed to integrate eqns. (4) and (5) are given in section 3.5 below. The equations used to calculate the various forces in the right hand side of eqn. (5) are formulated as follows (for technical details on the implementation of the various forces, the reader is referred to Users Manual included at the end of this report).

3.4.1 Viscous drag

The drag force acting on a bubble of frontal area S_b that moves at velocity $\vec{V}_b$ through a fluid of density ρ, is given as follows:

$$\vec{F}_D = \frac{1}{2} C_D \rho S \left| \vec{V} - \vec{V}_b \right| \left(\vec{V} - \vec{V}_b \right) \tag{6}$$

where $\vec{V}$ is the fluid velocity and C_D is the drag coefficient which is a function of the bubble Reynolds number. Since the bubble is assumed to be spherical, C_D can be readily determined from available experimental correlations. Using a compilation of such existing experiments, Munson et al. (1994), the variation of the drag coefficient for a sphere with the Reynolds number can be tabulated and all necessary values can be computed via suitable interpolations. It is important to point out that the drag force has been found to be the most important among the various forces in the right hand side of equation (4).

3.4.2 Force due to ambient pressure gradient

In a complex three-dimensional flow environment, the pressure sensed by the various sides of the bubble is not uniform but depends on the local pressure gradients in the flow. A simple and fast interpolation scheme is employed to estimate the net pressure force, denoted by F_P. As a general remark, we must say that this force produces two interesting effects on the average:

- since the draft tube is a pressure recovery system, in general pressure downstream of the bubble are greater than those upstream. This results in a net negative pressure force (opposing the propagation of the bubble with the flow)
- in swirling flows, this term causes bubble caught in the vortex core to move toward the center of the vortex and possibly coalesce

Examples of both the above can be found in the Results section of this report.

3.4.3 Force due to added mass effect

In order to account for the response of the fluid surrounding the bubble to acceleration, the so-called added mass effect, an additional force term is introduced as follows:

$$\vec{F}_{AM} = a\frac{4}{3}\pi r_b^3 \rho_{water} \frac{d}{dt}\left(\vec{V} - \vec{V}_b\right) \tag{7}$$

where r_b is the radius of the bubble and a is the added mass coefficient which for a sphere is equal to 0.5 (Newman, 1977). The velocity derivative along the three directions is computed via a first order accurate finite difference scheme, from current and stored bubble velocity values.

The numerical integration of the equations of motion (eqn. (4) can be greatly simplified and stabilized by moving the added mass force to the left hand side of the these equations. This amounts to substituting the mass of the bubble with a new effective mass that accounts for both the inertial mass and the added mass.

3.4.4 Buoyancy force

The net buoyancy force (effective bubble weight) acts in the vertical direction and is computed as follows:

$$\vec{F}_B = \frac{4}{3}\pi r_b^3 (\rho_{air} - \rho_{water})\vec{g} \tag{8}$$

where $\vec{g}$ is the gravitational acceleration. In the coordinate system used in our CFD simulations, $\vec{g} = (g,0,0)$. The density of the air in the bubble is computed (for this and all other purposes) from the ideal gas law and the local computed pressure.

3.5 Numerical integration of the equations of motion

The governing equations of bubble motion, eqns. (4) and (5), are integrated in time in a Lagrangian fashion. That is, unlike the governing flow equations which are solved on a fixed Eulerian mesh, the solution of eqns. (4) requires the calculation at every time step of both the bubble properties and spatial locations. This implies that any numerical scheme to be used for this purpose should consist of two components: i) a temporal integration scheme for advancing in time eqns. (4) and (5); and ii) an algorithm for searching and interpolating in space. As is the case with all our modeling choices in this work, the selection of an appropriate numerical scheme was guided by the need to balance computational efficiency and numerical accuracy.

3.5.1 Temporal integration scheme

Extensive numerical experiments with temporal integration schemes showed that schemes that are second order accurate and higher yield identical results for the bubble trajectories, provided that the time step is kept sufficiently small. However, schemes whose accuracy is higher that second order require either excessive memory (Euler type schemes) or significantly more computational time (Runge-Kutta, predictor-corrector and other multi-stage type schemes). Both of the above requirements can substantially increase the overall computational overhead, particularly when such schemes are employed to integrate in time the

trajectories of several thousands of air bubbles. Since we found no significant accuracy improvements with the use of a higher-than-second order approximation, the three-point, second-order accurate Euler explicit scheme was selected for integrating both eqns. (4) and (5):

$$\left(\frac{du_{bi}}{dt}\right)^{n+1} \approx \frac{3u_{bi}^{n+1} - 4u_{bi}^{n} + u_{bi}^{n-1}}{2\Delta t} \tag{9}$$

where n denotes the time level and Δt is time step. The time step in eqn. (10) is selected in a manner that guarantees numerical accuracy and stability while minimizes the computational resources required for carrying out spatial searches and interpolations. A module has been introduced in the code that pre-estimates[1] mean bubble traveling times along the three directions of every cell of the CFD computational grid. Consequently, the smaller of these traveling times is chosen as the time step (usually multiplied by a factor of 0.1-0.5, to increase accuracy and take into account inertia effects). This approach yields a very conservative time step estimate but has two major advantages: i) the time step is kept small enough for the temporal integrator to be accurate and stable; and ii) it guarantees that the spatial position of a given bubble at the new time level will be in the close neighborhood of its current position.

A small example can illustrate clearly the speedup achieved by selecting the time step as described above. Assuming that the new position of the bubble will be within, say, 4 computational cells from the old one, the required search area consists of $(4+1+4)^3=729$ cells (four cells upstream, the current cell and four cells down stream, for all three spatial directions). If our estimation involves a neighborhood of 10 computational cells, we get a total of $(10+1+10)^3$ =9,261 cells to be searched. Arbitrarily defined, user-specified time step requires a searching area that spans 10-15 cell neighborhoods in every direction. The time step selected using the above procedure allows the use of just 1 cell neighborhoods, which implies that the total number of grid cells to be scanned is $(1+1+1)^3=27$. Since the search algorithm takes up more that 75% of the total CPU usage of the model, it is obvious that a speedup of 0.75X(9,261/27)=250 per time step is achieved through this technique. Of course the final speedup of the model is reduced by a factor 10-15 because the smaller time step means increased number of time steps required for the completion of each trajectory. Still, a significant overall speedup of approximately 15 has been observed.

It is important to point out that the code is constructed in such a way, that if the initial 1-cell-neighborhood fails, then all of the computational domain is searched. This happens very rarely, however, and usually only for newly injected bubbles. The effort for these newly injected bubbles is still very small, because bubbles are injected near the first sections of the computational grid (near the inlet plane of the draft tube) thus the algorithm locates the corresponding cells without having to search but a small number of cells. The actual mechanism that this locating takes place is described in the next section.

[1]This is done only once, in the beginning of each run

3.5.2. Spatial search and interpolation algorithm

In order to be able to estimate the local flow conditions around the bubble we need to pinpoint the location of the bubble in the computational flow field. This is not an easy task, since computational grids for draft tubes are in general curvilinear, skewed, stretched and very irregular. The technique used to find the grid cell that the bubble is in is based on an equality of volumes principle. Each of the grid cells is subdivided in six tetrahedra that span the original volume. The volume of each one of these tetrahedra is computed[2] using a simple analytic geometrical relationship and stored. Subsequently, during regular execution of the program, the searching algorithm assumes that the center of the bubble is in every one of these tetrahedra and defines four new tetrahedra for each one of the initial ones. The four vertices of the new sub-tetrahedra correspond to three vertices of the original tetrahedron and the center of the bubble. The sum of the volumes of these new four tetrahedra will be equal (within some accuracy depending on roundoff error) to the original tetrahedron volume, if and only if the center of the bubble is within this tetrahedron (figure 3).

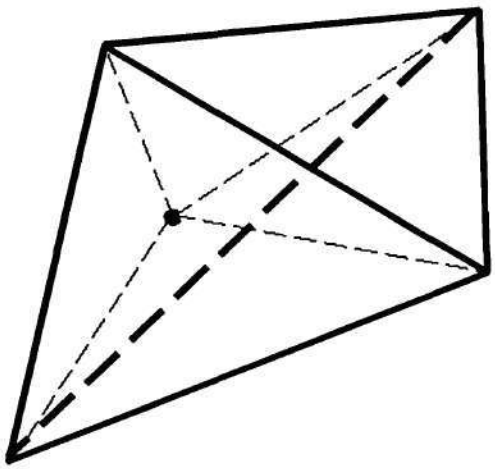


Figure 3. Schematic of the technique used to locate the center of a bubble in space

When this is satisfied, we declare the center of the bubble to be in that cell and interpolate the values of the variables from the eight grid nodes defining that grid cell. An inverse distance formula, with an exponent of 3.5 is used for the interpolation:

$$\Phi_{bubble} = \frac{\sum_{nd=1}^{8} \Phi_{nd} d_{nd}^{3.5}}{\sum_{nd=1}^{8} d_{nd}^{3.5}} \qquad (10)$$

where d_{nd} is the distance of the center of the bubble from every cell vertex.

[2] This is done only once, in the beginning of each run

For very skewed grids, the above search algorithm might fail due to roundoff errors in the calculation of the cell volumes. Our experience so far has shown that this occurs very rarely. In the rare occasion that this happens, the user is provided with a hard-coded constant that can be altered (increased slightly) to accommodate these roundoff errors.

4. Estimation of Momentum Source Terms

Once the distribution of bubbles in the draft tube has reached a steady state, the next step is to compute the momentum deficit these bubbles actually impose on the flow field and formulate that deficit it terms of momentum source terms for the RANS equations. This procedure involves the following steps:

a) Estimate the force each bubble is exerting on the surrounding flow, which is equal to the drag of the bubble:

$$\vec{F}_D = \frac{1}{2} C_D \rho S \left| \vec{V} - \vec{V}_b \right| \left(\vec{V} - \vec{V}_b \right) \tag{11}$$

b) Scale this force estimate according to the velocities along the three cartesian directions. Possible scaling required by specific CFD solvers can be incorporated in a straightforward manner at a postprocessing step.

c) Identify, for every bubble, the cell that it lies in and add the contributions of that bubble to a summation registers that carry the combined contributions of all the bubbles that are in that cell. At the end of this step, we end up with a distribution of momentum deficits that spans our flow field, this does not imply however that all the computational cells of our original CFD grid actually carry such terms, since cells that contain no bubbles will have zero contributions.

d) Apply a smoothing operator to the raw source terms distribution. A simple Laplacian smoothing function of user specified weighting concludes the procedure. This step is deemed necessary for two reasons. First, obvious computational restrictions limit the number of simulated bubbles to a few thousand. It is therefore impossible to achieve a bubble distribution that resembles the one realized either experimentally or at the actual powerplant. By smoothing the raw bubble contributions we are achieving an overall more realistic distribution, since, in the computations it is quite common to have two nearby cells being occupied by bubbles and the adjacent cell between them empty, something that is rather unlikely to happen in realistic experiments (at least for cells of the size that is usual in CFD computations). By smoothing the source terms, this middle, unoccupied, cell will not have a zero contribution anymore, but will be specified some (depending on user specified parameters controlling the amount of smoothing applied) source term magnitude, closely correlated with those of the neighboring cells. The second reason that smoothing is necessary is that CFD solvers, that are to re-compute the flow field taking into account these new source terms, behave better, both in terms of convergence but also in terms of quality of the solution, when the forcing applied is more or less continuous. It is a well know fact that practically all numerical methods suffer in regions of severe discontinuities, and applying the highly scattered raw distribution of source terms is expected to yield results with poor accuracy characteristics, if convergence is achieved at all.

5. Results and discussion

The method developed herein has been applied to simulate bubble trajectories and DO transfer for one sample case, with data provided by the CFD team of Voith Hydro, Inc. This data field corresponds to a typical CFD draft tube solution, obtained using the commercial RANS solver TASCFLOW. The 31 blocks of the grid configuration for this draft tube are presented in figure 4.

As already discussed, the shape of the aeration openings has been described in an approximate manner. For this sample test case, a set of 20 openings, distributed along four radii of the ring, and set to be rotating elements, have been used. This configuration would resemble aeration from the blades, a technique that is often used in practice.

The program was run until an equilibrium in the number of bubbles in the computational domain was reached, i.e. when the number of bubbles exiting the draft tube minus the number of bubbles entering the draft tube was constant over an adequate time interval. The total number of bubbles tracked for equilibrium was approximately 2,500. The computer time needed for this run was about 6 CPU hours[3]. The reported time corresponds to a high-end Silicon Graphics Origin 2000 server. The user can adjust the total number of bubbles to the speed of the available computer by appropriately adjusting the number of time steps between two successive bubble-injection events. For this test, the oxygen concentration of the water downstream of the runner was set equal to 1 mg/L. The saturation concentration is assumed to be 46.2 mg/L In subsequent figures, the bubble sizes have been enlarged for clarity.

In figure 5, the evolution of the bubble distribution in this flow field is presented, for a few typical time steps of the computation. The bubbles are depicted as constant size circles in this figure, a technique that allows for fast representation of results, in stationary or animated fashion, but fails to convey the fact that the bubbles are actually diminishing as they pass their air to the water.

Figure 6 represents one particular time step of the computation, after post-processing the bubble information provided by the program. Here the bubbles are depicted as shaded spheres, with radii that are proportional to the actual bubble size, thus allowing for the diminishing effect to be demonstrated.

Figure 7 illustrates one additional feature that is incorporated in the technique, that is bubble tagging and color-coding. Each individual injection port is assigned a unique identification number, and this number is inherited by all the bubbles injected from that port. In this manner we can track and color-code all the bubbles injected by a port, or by a set of ports as shown in this figure. This feature is useful when evaluating the effectiveness of particular

[3] We should note here, that the currently described version of the code displays a somewhat slower performance than the pre-existing single-grid bubble tracking program. This is attributed to the additional effort necessary for the determination of the block that each bubble is in.

aeration strategies.

In figure 8, an iso-surface of the x-direction momentum source term distributions is presented, before any smoothing is applied. Two observations we can make from this figure, is that the overall distribution of the source terms follows more or less the actual distribution of the bubbles and that the surface is of particularly irregular shape, a fact that implies sharp discontinuities within the domain.

In figure 9, the same value for the x-direction momentum source term distribution shown in figure 8 is presented, only in this case, a moderate amount of smoothing has been applied. We can see that the overall distribution remains relatively unchanged, however the shape of the iso-surface is much smoother and diffused. Although, up to this time, no testing of a CFD solver with the incorporation of these source terms has been performed, it is expected that the capability for smoothing the original source term distribution will render such an application feasible.

The smoothing effect is better depicted in figures 10 and 11, where a 2-D slice through the elbow of the draft tube is presented. Figure 10 corresponds to the raw source term distribution whereas figure 11 is the smoothed one, the improvement of the contours as far as their continuity characteristics is quite apparent from these two figures.

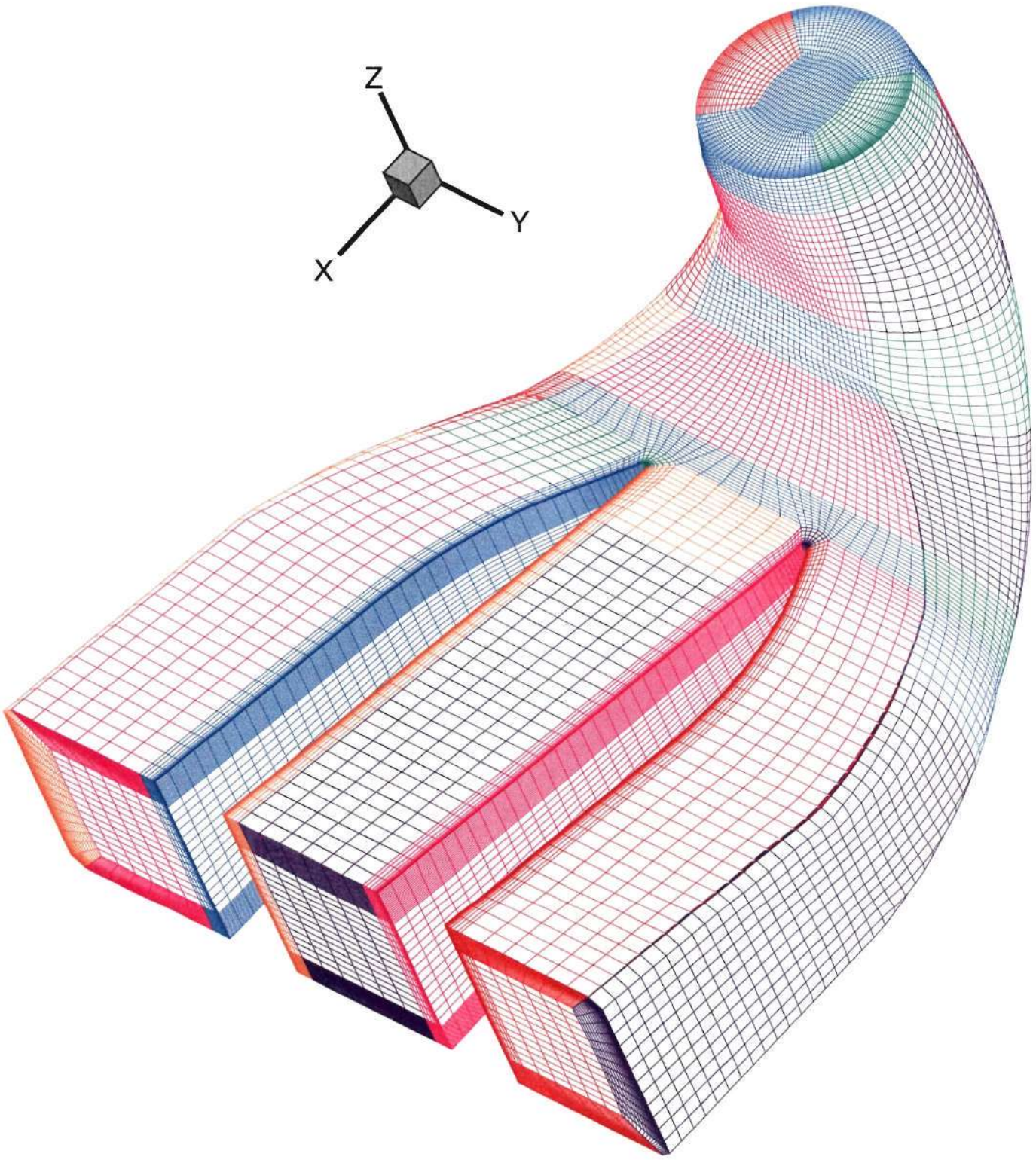


Figure 4. General view of the 31-grid block, 3-bay draft tube tested. The various blocks are color-coded for clarity

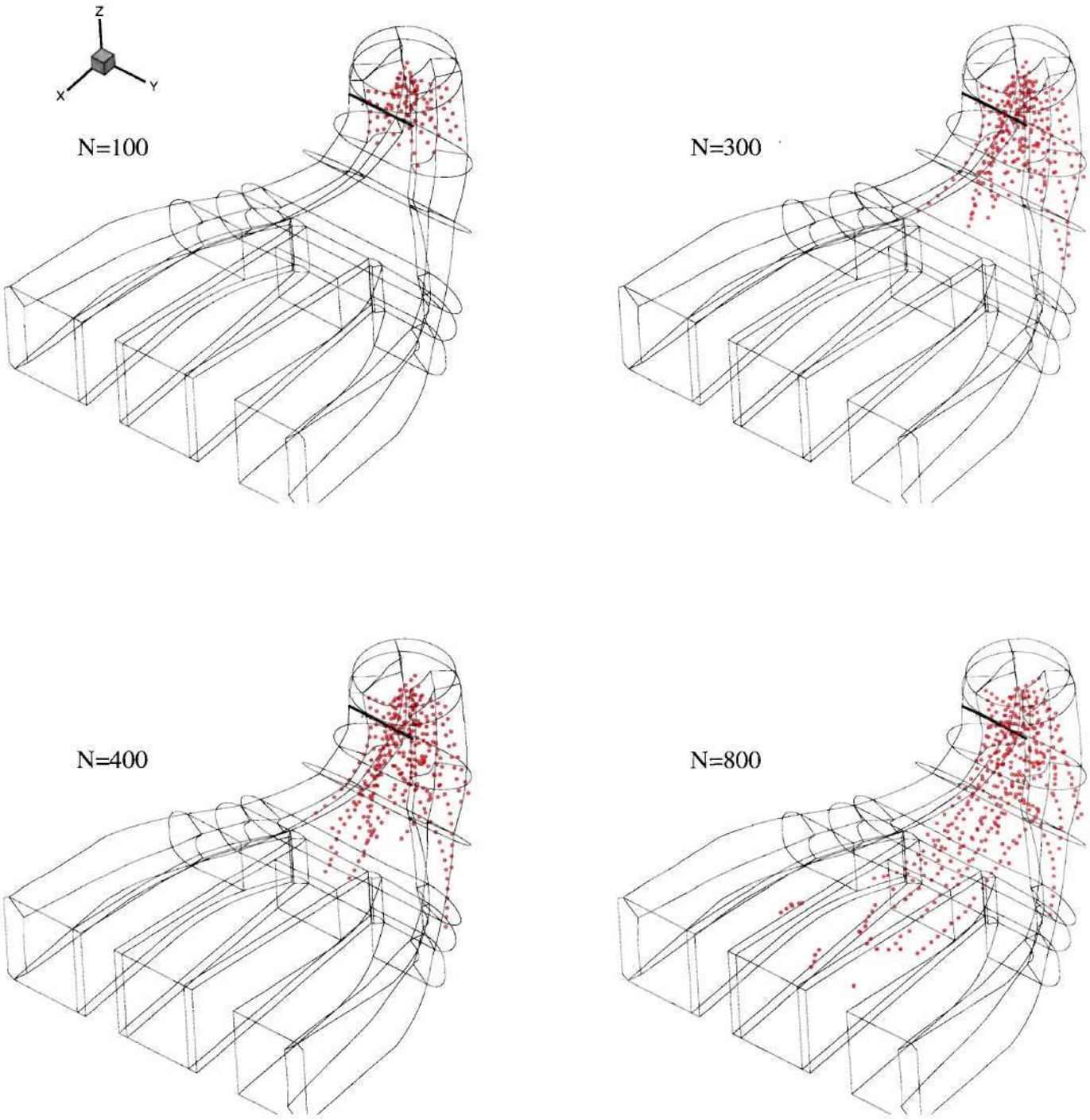


Figure 5. Evolution of the bubble distribution for 4 representative time steps. Bubbles are respesented as constant pitch circles

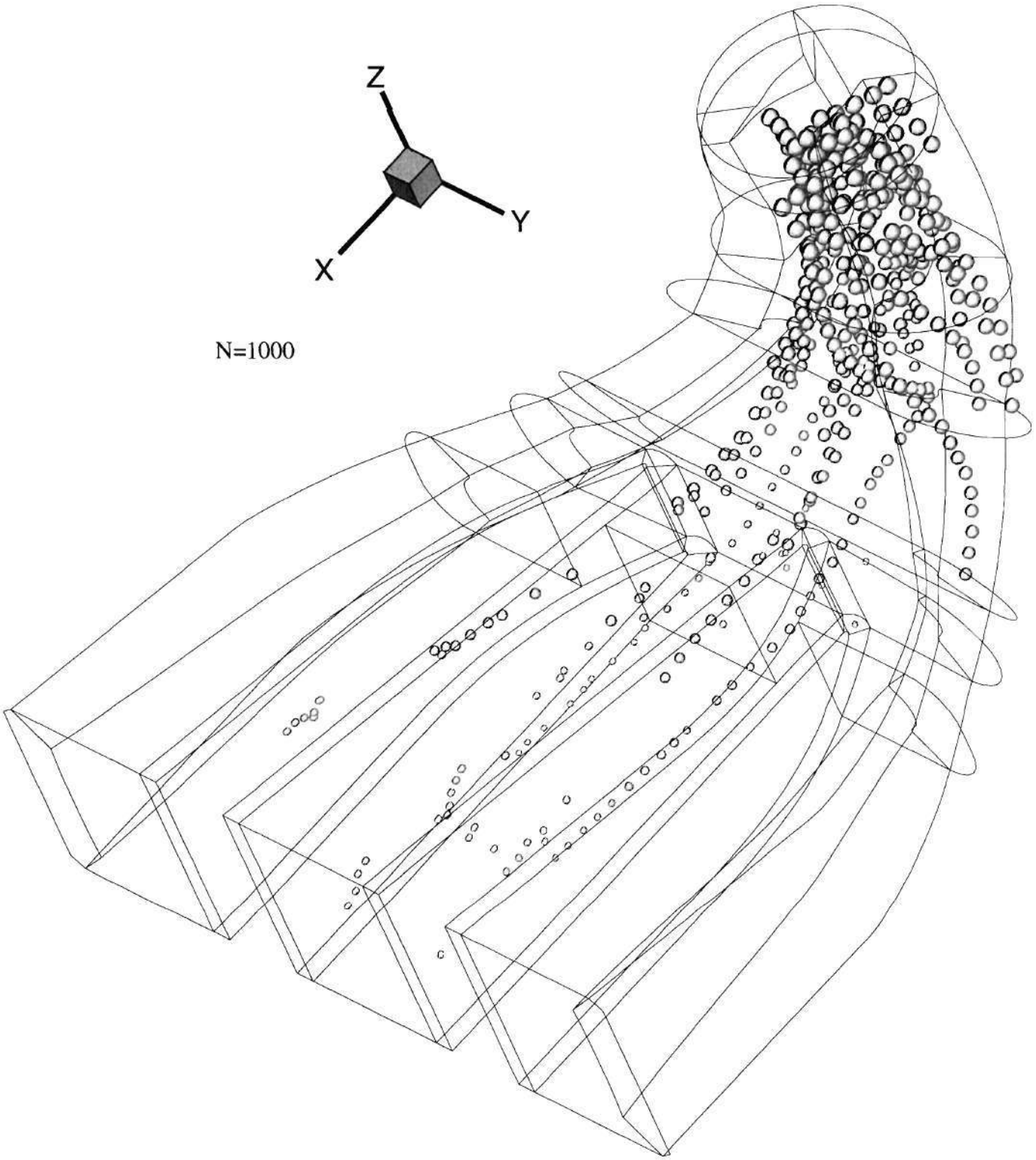


Figure 6. Evolution of the bubble distribution for a representative time step. Bubbles are respesented as spheres of radius proportional to the bubble size

Figure 7. Bubble distribution for a typical time step. Color-coding according to injection port allows the determination of the effectiveness for various aeration techniques

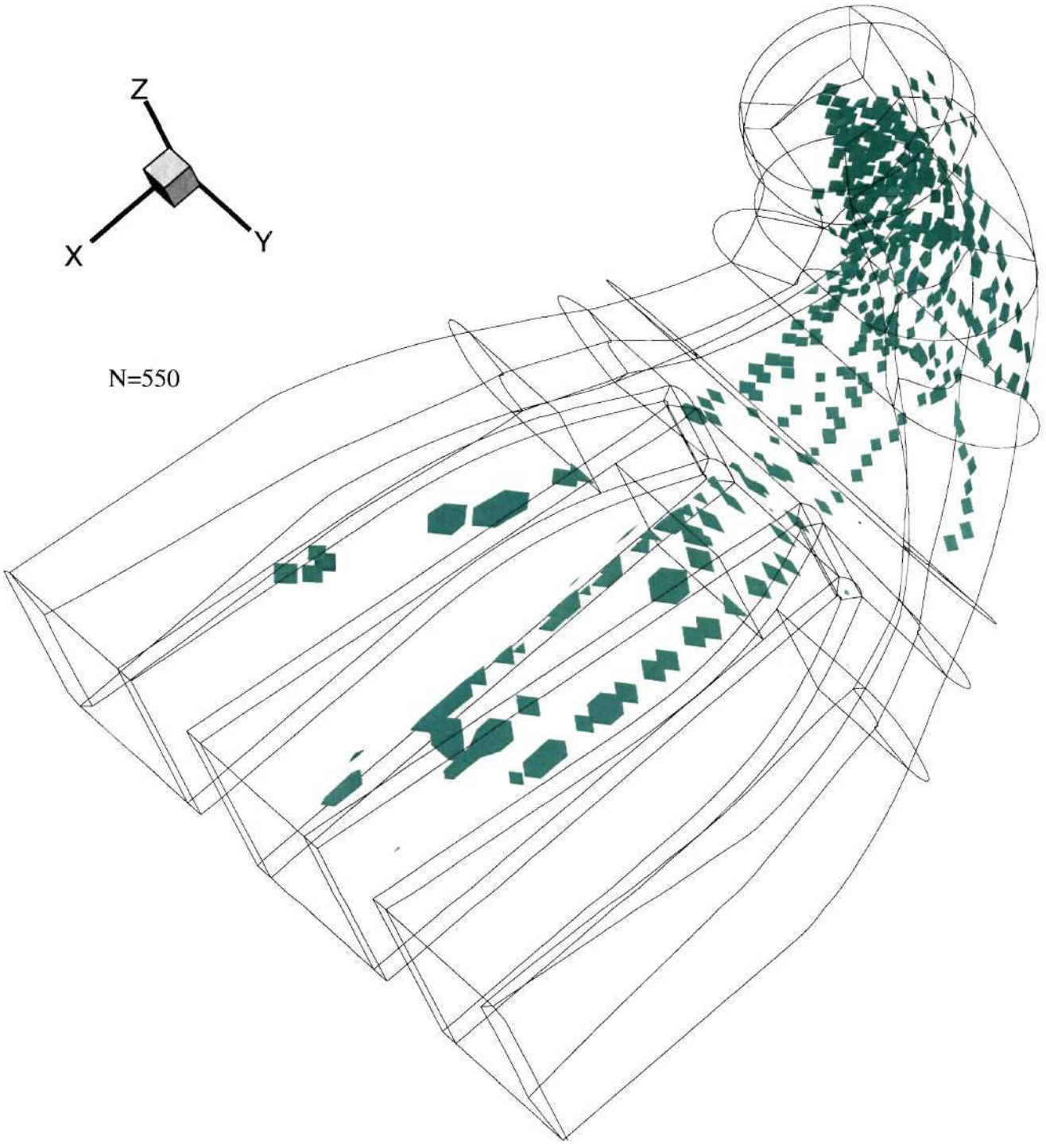


Figure 8. Characteristic iso-surface of the distribution of x-direction momentum source terms

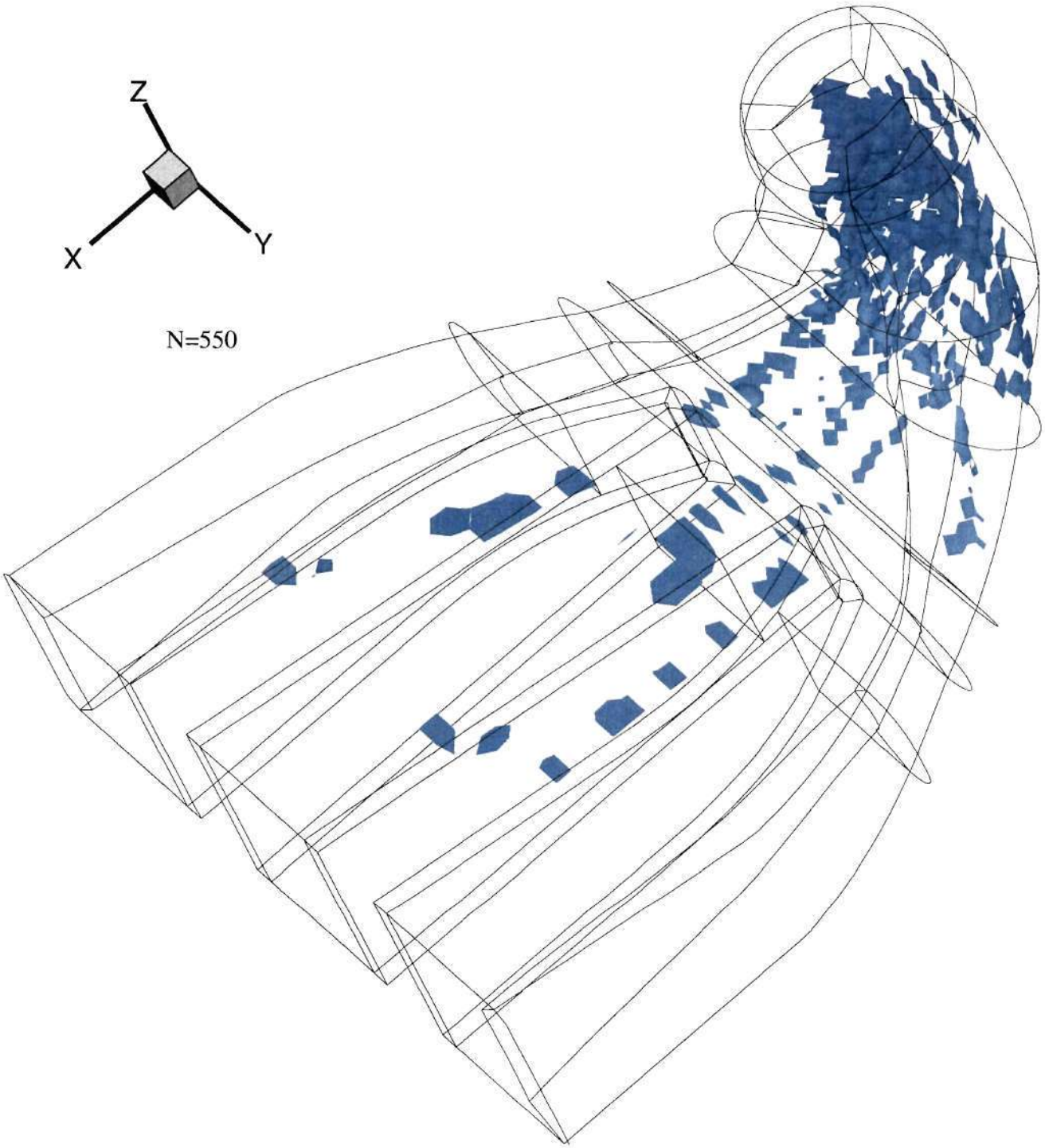


Figure 9. Characteristic iso-surface of the distribution of x-direction momentum source terms, as depicted in figure 8, after smoothing

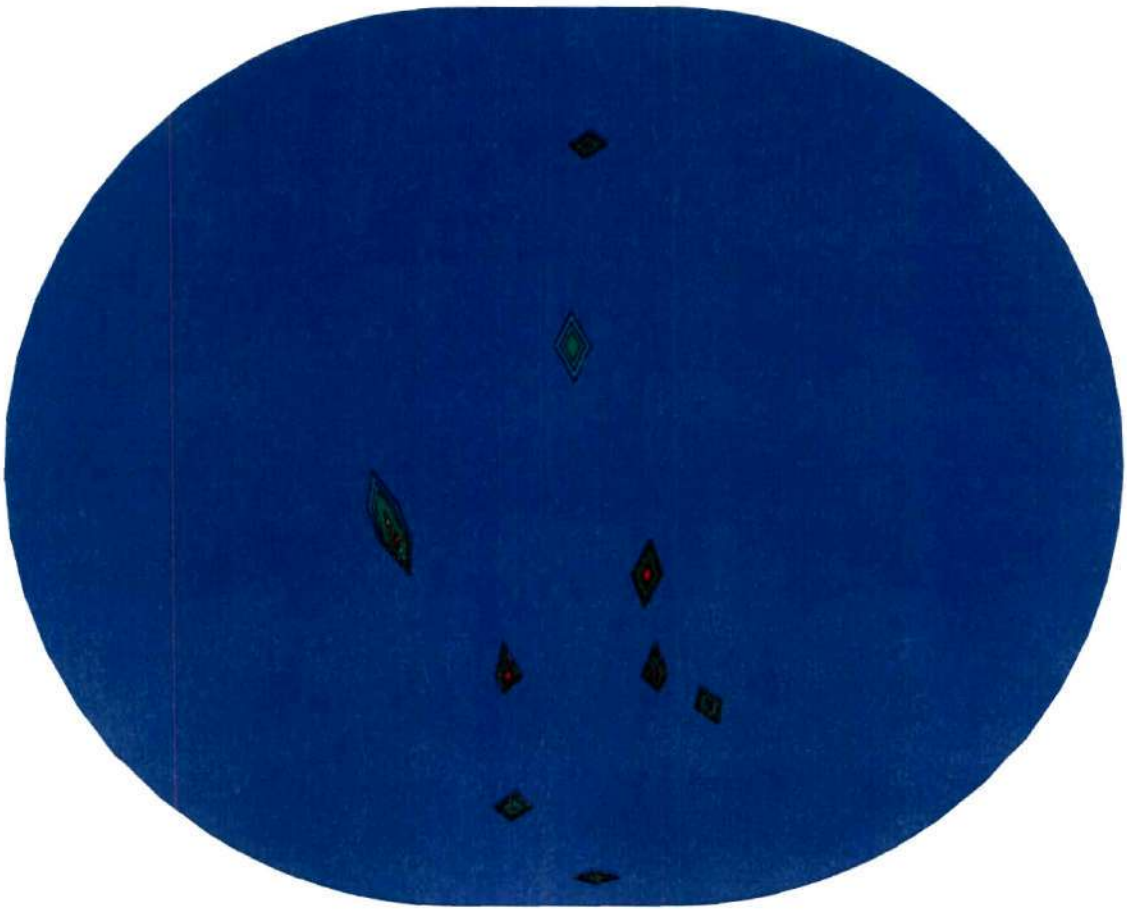


Figure 10. Contours of x-momentum source terms distribution for a two-dimensional section of the draft tube in the elbow region

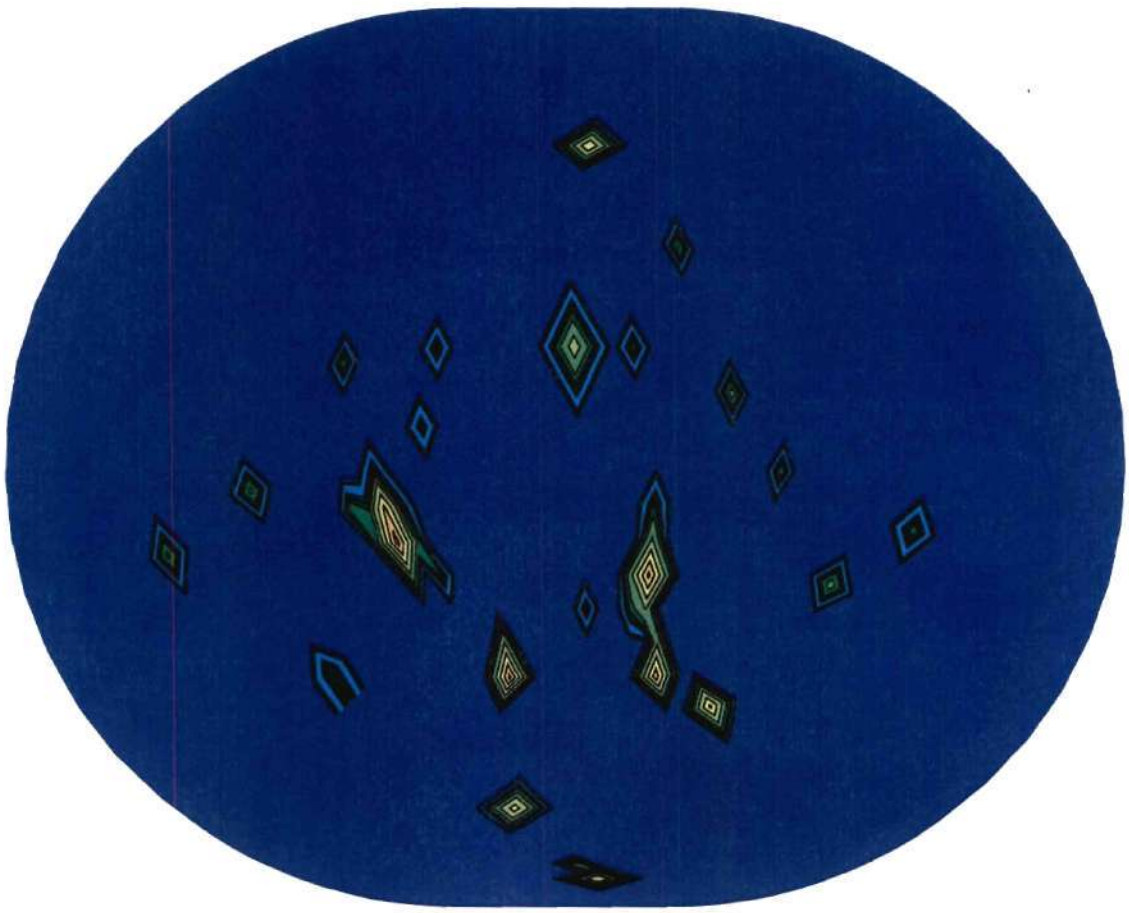


Figure 11. Contours of x-momentum source terms distribution for a two-dimensional section of the draft tube in the elbow region, as depicted in figure 10, after smoothing

6. Summary and conclusions

A three-dimensional numerical model was developed for tracking individual bubble trajectories, computing DO transfer, and estimating momentum deficit source terms caused by the bubbles, in autoventing hydroturbine draft tubes. The equations governing bubble motion are formulated in Lagrangian form and integrated in time through a precomputed, via a separate CFD calculation, turbulent flow environment. Forces due to viscous drag, ambient pressure gradient, added-mass effects, gravity, and buoyancy comprise the source terms of the bubble equations of motion. The model accounts for bubble breakup, bubble coalescence, and DO transfer from the bubbles to the water, under the following assumptions:

- the flow inside the draft tube is steady and not immediately affected by the motion of the air bubbles
- the statistical mean bubble shape is spherical;
- bubble split-up and coalescence take place only in a binary fashion; and
- the capacity of the water to dissolve DO at any instant time is not affected by the amount of DO that was dissolved at earlier times.

The model was applied to simulate bubble motion and DO transfer for one test case. The computed results demonstrate the potential of the proposed approach as a powerful engineering tool for understanding the highly non-linear dynamics of bubble motion and refining air-injection strategies.

At its current state of development, the model can be used to provide only general qualitative trends. A number of modeling refinements as well as detailed validation studies with experimental measurements are necessary in order to enhance its quantitative accuracy. Future work should focus on: i) detailed quantitative validation of the flow solver over a range of powerplant operating conditions; ii) incorporating a transport equation to account for history effects on the DO concentration of the water; iii) developing physically sound estimates for the initial bubble size and the bubble break-up delay time; and iv) obtaining detailed DO data to validate and fine-tune the mass-transfer module of the model.

7. References

Baird M. H. I., Rohatgi A., "Mass transfer from discrete gas bubbles in a reciprocating plate column", The Canadian journal of Chemical Engineering, 67, 1989, p 682

Brice T. A., Cybularz J. M., "Air admission effects on hydraulic turbines", FED-Vol. 136, ASME 1992, p 121

Cho J. S., Wakao N., "Determination of liquid-side and gas-side volumetric mass transfer coefficients in a public column", Journal of Chemical Engineering of Japan, 21, 6, 1988, p 576

Cuenca-Alvarez M., Baker C. G. J., Bergougnou M. A., "Oxygen mass transfer in bubble columns", Chemical Engineering Science, 35, 1980, p 1121

Daniil E. I., Gulliver J. S., "Temperature dependance of liquid film coefficient for gas transfer", Journal of Environmental Engineering, 114, 5, 1988, p 1224

Daniil E. I., Gulliver J. S., "Water quality impact assessment for hydropower", Journal of Environmental Engineering, 117, 2, 1991, p 179

Daniil E. I., Gulliver J. S., "Influence of waves on air-water gas transfer", Journal of Environmental Engineering, 117, 5, 1991, p 522

Domgin J. F., Huilier D., Burnage H., Gardin P., "Coupling of a Lagrangian model with a CFD code: Application to the numerical modeling of the turbulent dispersion of droplets in a turbulent pipe flow", Journal of Hydraulic Research, 35, 4, 1997, p 473

Fu T. C., Shekarriz A., Katz J., Huang T. T., "The flow structure in the lee of an inclined 6:1 prolate spheroid", Fluid Mech., 269, 1994, p 79

Gulliver J. S., Arndt E. A., "Interfacial support in river- reservoir systems", FED- Vol. 143/HTD-Vol. 232, ASME 1992, p 77

Gulliver J. S., Halverson M.J., "Gas transfer and secondary currents in open channels", Water Forum '86, p 1056

Gulliver J. S., Halverson M.J., "Measurements of large streamwise vortices in an open-channel flow", Water Resources Research, 23, 1, 1987, p 115

Gulliver J. S., Oakley B. T., Semmens M. J., "A new in-stream aerator", Hydraulic Engineering '93, p 2165

Gulliver J. S., Rindels A. J., "Measurement of air-water oxygen transfer at hydraulic structures", Journal of Hydraulic Engineering, 119, 3, 1993, p 327

Gulliver J. S., Stefan H. G., "Stream productivity analysis with dorm -I Development of computational model", Water Res., 18, 12, 1984, p 1569

Gulliver J. S., Sundquist M., Voigt R. L., Jr., Hibbs D. E., "The Brasfield hydroelectric project A model-prototype comparison", Waterpower '95, San Francisco CA, p 2361

Gulliver J. S., Wilhelms S. C., " Water quality enhancement technology for river-reservoir systems", Proc. Natl. Conf. Hydraul. Eng., 1994 , p 1331

Hadjerioua B., Eldredge T. V., Mobley M. H., "Reservoir oxygenation by oxygen diffusers",Int. Water Res. Eng. Conf. Proc. 1995, New York NY, p 1451

Harshbarger E. D., Mobley M. H., Brock W. G., "Aeration of hydroturbine discharges at Tims Ford dam", Waterpower '95, 1995, San Francisco CA, p 11

Herringe R. A., Davis M. R., "Structural development of gas-liquid mixture flows", J. Fluid Mech., 73, 1, 1976, p 97

Hibbs D. E., Gulliver J. S., "Prediction of dissolved gas supersaturation below spillways", Waterpower '95, 1995, San Francisco CA, p 173

Hesketh R. P., Etchells A. W., Russell T. W. F., "Bubble breakage in pipeline flow", Chemical Engineering Science, 46, 1, 1991, p 1

Hesketh R. P., Etchells A. W., Russell T. W. F., "Experimental observations of bubble breakage in turbulent flow", Ind. Eng. Chem. Res., 30, 1991, p 835

Hughmark G. A., "Drop breakup in turbulent pipe flow", AIChE journal, 17. 4, 1971, p 1000

Jun K. J., Jain S. C., "Oxygen transfer in bubbly turbulent shear flow", Journal of Hydraulic Engineering, 119, 1, 1993, p 21

Lewis D. A., Davidson J. F., "Mass transfer in a recirculating bubble column", Chemical Engineering Science, 40, 11, 1985, p 2031

Luo H., Svendsen H. F., "Theoretical model for drop Breakup in turbulent dispersion", AIChE Journal, 42, 5, 1996, p 1225

Maxworthy T., "Bubble rise under an inclined plate", J. Fluid Mech., 229, 1991, p 659

Michaelides E. E., "Review - The transient equation of motion for particles, bubbles, and droplets", J. Fluid. Eng., 119, 1997, p 233

Mobley M., Tyson W., Webb J., Brock G., "Surface water pumps to improve dissolved oxygen content of hydropower releases", Waterpower '95, 1995, San Francisco CA, p 21

Motarjemi M., Jameson G. J., "Mass transfer from very small bubbles - The optimum bubble size for aeration", Chemical Engineering Science, 33, 1978, p 1415

Munson B. R., Young D. F., Okiishi T.H., "Fundamentals of fluid mechanics", John Willey and Sons, 1994

Neti S., Mohamed O. E. E., "Numerical simulation of turbulent two-phase flows", Int. J. Heat and Fluid Flow, 11, 3, 1990, p 204

Newman J. N., "Marine Hydrodynamics", The MIT Press, 1977, p 32

Rindels A. J., Gulliver J. S., "Air-water oxygen transfer at spillways and hydraulic jumps", Waterforum '86, p 1041

Roberts G. O., Kornfeld D. M., Fowlis W. W., "Particle orbits in a rotating liquid", J. Fluid Mech., 229, 1991, p 555

Rowe P. N., "Drag forces in a hydraulic model of a fluidised bed- part II", Trans. Instn. Chem. Engrs, 39, 1961, p 175

Shinnar R., "On the behavior of liquid dispersions in mixing vessels", J. Fluid Mechanics, 10, 2, 1961, p 259

Su W., Tao B., Xu L., "Three-dimensional separated flow over a prolate spheroid", AIAA Journal, 31, 11, 1993, p 2175

Tamburrino A., Gulliver J. S., "Free-surface turbulence measurements in an open-channel flow", FED-Vol. 181, ASME 1994, p 103

TASCFLOW manual, 1997

Ventikos, Y., Sotiropoulos, F., and Patel, V. C. "Prediction of Turbulent Flow through a Hydroturbine Draft-Tubes Using a Near-Wall Turbulence Closure," Proc. of XVII IIAHR Symp. on Hydraulic Machinery and Cavitation (Cabrera, Espert, and Martinez, Eds.), vol. I, pp. 140-149.

Wace P. F., Burnett S. J., "Flow patterns in gas-fluidised beds", Trans. Instn Chem. Engrs, 39, 1961, p 168

Wahl T. L., Young D., "Dissolved oxygen enhancement on the Provo river", Waterpower '95, 1995, San Francisco CA, p 1

Waldrop W. R., "Overdview of autoventing turbine technology development project", Proc. National Conf. On Hydraulic Engineering, 1995, New York N.Y., p 257

Weiss P. T., Oakley B. T., Gulliver J. S., Semmens M. J., "The performance of a vertical fiber membrane aerator", FED-Vol. 187, ASME 1994, p 59

Wells M. R., Stock D. E., "The effects of crossing trajectories on the dispersion of particles in a turbulent flow", J. Fluid Mech., 136, 1983, p 31

Wetzel J. M., Voigt R. L., Gulliver J. S., Georgiou- Foufoula E., Stefan H. G., Arndt R. E. A., "The benefits of applied research to hydropower development", Proceedings of the American Power Conference, 1995, p 472

APPENDIX A: Users Manual

In the sequel, we shall describe the structure and operation of the Fortran code created. Text that appears with `Courier fonts` corresponds to file names, code constants, variables and subroutines and in general to elements of the actual computer program. The files necessary for a computation are:

- the source Fortran code `bubble.f`
- the include common block file `com-bubble`
- the executable obtained from compiling the source Fortran code `bubble.f`
- the include common block file `com-bubble`
- the main data file `CONTROL`
- a set of grid specification files (names defined in main data file `CONTROL`)
- a set of solution specification files (names defined in main data file `CONTROL`)
- post-processing tools and programs, like the `multi-id-bubbles.f` code

Description of the code

The result of the research effort described so far is the Fortran computer code `bubble.f`. The code has been tested in various platforms, from personal computers and workstations to supercomputers, for portability and performance. The hardware and software requirements for a successful execution of the code are:

REQUIREMENTS	Minimum	Suggested
CPU[4]	RISK processor or PENTIUM 200 MHZ	Last generation RISK processor (R8K, Alpha, R10K, Ultra) or PENTIUM III 500 MHZ
MEMORY	256 Mbytes	512 Mbytes
HARD DISK SPACE	100 Mbytes per draft tube configuration	200 Mbytes per draft tube configuration
OPERATING SYSTEM	UNIX or WINDOWS NT	UNIX or WINDOWS NT
FORTRAN COMPILER	ANSI Fortran or newer	ANSI Fortran or newer

Table 1. System requirements

[4] Although the program will run on medium power, Pentium based, personal computers, it is best suited for high-end Unix workstations, where a few thousand bubbles can be tracked simultaneously within reasonable time.

A compromise between modularity/adaptability and execution speed has been made. More specifically, core parts of the algorithm that are extremely time consuming and are not bound to serious updates in future versions, are quite efficiently but rather obscurely coded. On the other hand, most of the physical modeling part is very easy to adapt and upgrade.

The global variable approach has been used during the construction of the code, meaning that most variables are globally addressable throughout the code. To facilitate this, the use of a single include file containing all the variable definitions has been implemented and call from every subroutine of the code.

A single data file (named `CONTROL`) is used to specify all user supplied data to the code. The structure of this file is described in the sequel. An average Fortran programmer can very easily alter the appropriate read statements in the `readfield` subroutine to enable the code to input differently formatted data.

The code produces a number of output files, that are described here:

`XYZ.dat`
A TECPLOT format file containing the grid blocks, useful for postprocessing graphics and animations.

`EVERYTHING.dat`
A TECPLOT format file containing the grid blocks, along with the CFD solution, useful for verification and postprocessing.

`ANIM1-***.plt`
Series of TECPLOT format files, written at user specified time intervals, containing the coordinates of the center of each bubble, useful for animations.

`RESULTS-***.plt`
Series of TECPLOT format files, written at user specified time intervals, containing the coordinates of the center of each bubble, the bubble identity (correlated to the injection port), the bubble radius and the amount of air in the bubble, useful for animations and statistical postprocessing.

`SOURCES.dat`
A TECPLOT format file, containing the grid blocks, along with the distribution of source terms, for the three directions, raw and smoothed, useful for prostprocessing.

`SOURCES-RES`
A serial ascii file, containing the distribution of source terms, for the three directions, raw and smoothed, to be used (after possible scaling) as input to a CFD RANS solver for the computation of the coupled water-air flow field.

Flowchart

A schematic representing the flow of the execution of the code is presented. Most of the modules presented in this diagram correspond to real code modules (or set of modules).

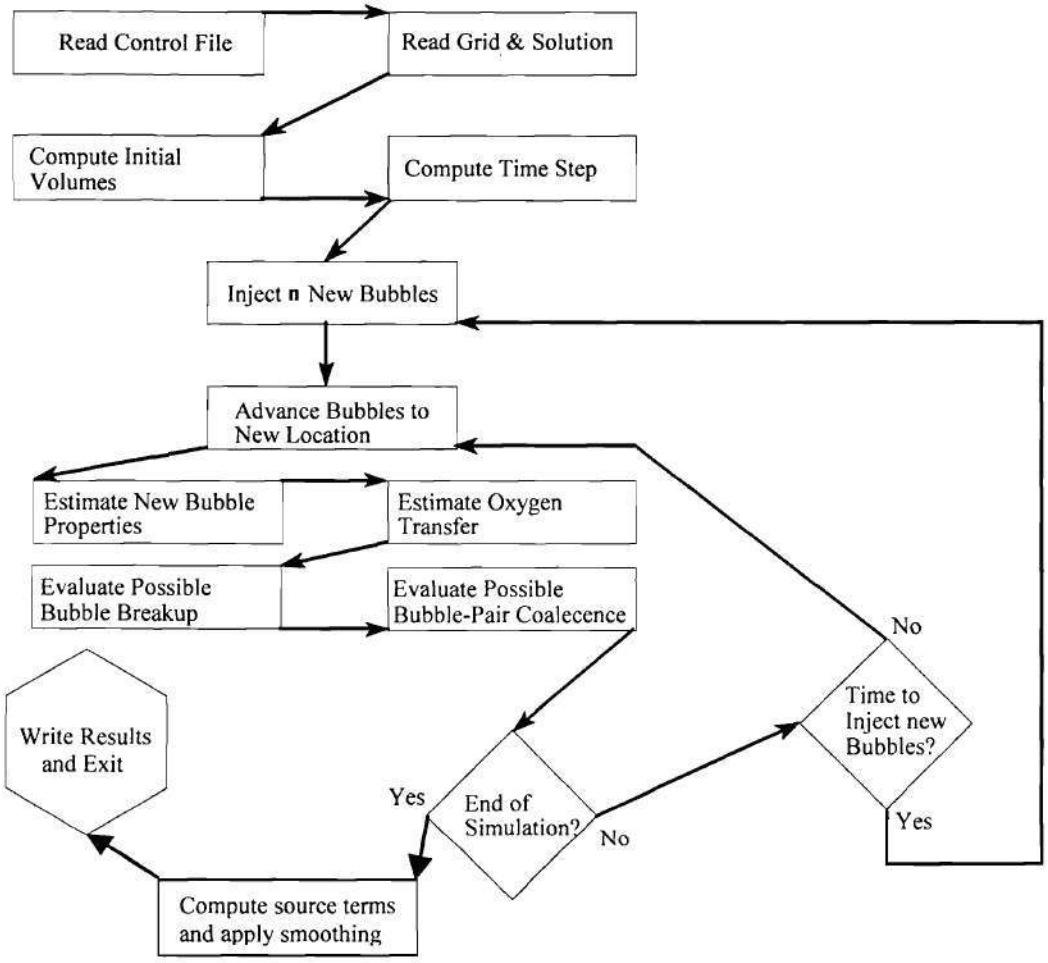


Figure 12. The code flowchart

The subroutines of the code

`program Virtual Bubble`

The main program calls a few preparatory subroutines and the subroutines `traject` (which does the bubble tracking) and `sources` (which computes the momentum source terms).

`subroutine cnstnts`

This subroutine specifies the values of all the important constants for the code. The Metric Unit System is used for all dimensional constants.

`subroutine readfield`

This subroutine reads the grid and the flowfield as computed from a CFD program.

`subroutine initpos`

This subroutine specifies the initial position of the bubbles, i.e. their points of entry in the draft tube. If the unsteady option is activated for an inlet boundary, this boundary is marched in time in an rotating fashion.

`subroutine traject`

This subroutine is the core of the code. It integrates the equations of motion for the bubbles in the draft-tube, propagates the bubble and performs all the necessary checks for the evolution of the bubbles

`subroutine locate(iconti)`

This subroutine finds in what cell of the computational mesh the center of the bubble is.

`subroutine ambient`

This subroutine determines the local conditions the bubble is sensing in its current location.

`subroutine march`

This subroutine advances the bubble to the next location along its trajectory. "Advances" means time marching integration of all three equations of motion.

`subroutine prepare`

This subroutine precomputes main cell volumes for faster execution of subsequent steps.

`subroutine inivol(xf1,xijk,xi1jk,xij1k,xijk1,`

This is the subroutine where the actual volumes are precomputed

`subroutine compdt`

This subroutine evaluates the time step to be used for the simulation.

`subroutine forces`

This subroutine computes and adds up all the forces exerted on the bubbles.

`subroutine physics`

This subroutine estimates the various physical properties of the bubble.

```
subroutine shifter
```

This subroutine properly shifts all bubble-related arrays to make space for the newly-to-be-injected bubbles.

```
subroutine swapthem
```

This subroutine investigates the state of each bubble (existent or inexistent) and rearranges all bubble-related arrays to carry only existent bubbles.

```
subroutine cdreyn(velrel)
```

This subroutine computes the bubble Reynolds number and the corresponding drag coefficient.

```
subroutine create_br(gnew,rnew,ibr,ibgive)
```

This subroutine arranges the new bubble, created from bubble breakup, in its temporary arrays.

```
subroutine newbubs(ibroken)
```

This subroutine adds the bubbles created by breakup to the main bubble arrays.

```
subroutine complamda
```

This subroutine computes the aeration ratio of the flow, lamda (λ).

```
subroutine geom
```

This subroutine pre-computes the directions of the grid cells, to speed up subsequent force computations.

```
subroutine dimensions
```

This subroutine scales the CFD variables (velocity, geometry, etc) to prototype scale.

```
subroutine pres_scl_dt
```

This subroutine re-dimensionalizes the pressure field, as given by the CFD solution, to prototype scale.

```
subroutine walls
```

This subroutine examines if a particular bubble has exited the domain, or has formed an air-pocket on a solid wall.

```
subroutine sources
```

This subroutine computes the momentum source terms for the three cartesian directions and for every grid cell and subsequently applies a smoothing operator to those source terms.

Variables and Constants

Throughout the construction of the code, effort has been made to have self-explanatory names of variables. A list of the most important program variables and constants, along with their meaning and significance follows. Arrays and matrices are specified as such and the role of the indices is explained. The Fortran naming convention (all variables are real except those starting with I,J,K,L,M,N) has been followed.

`initnum`
total number of bubbles to be injected at every injection-enabled time step

`xinlet(*),yinlet(*),zinlet(*),`
position of every bubble injection opening

`inlettype(*)`
type of opening (0 for stationary, 1 for opening rotating with the runner)

`radorif(*)`
initial radius of bubble

`cfsair(*)`
airflow rate in m^3/s

`iturbul`
type of turbulence model (0 for k-ϵ, 1 for k-ω)

`xlamda`
airflow rate/(waterflow rate + airflow rate)

`ububble(mgblb),vbubble(mgblb),wbubble(mgblb),pbubble(mgblb),xkbubble(mgblb)`
velocity components, pressure and turbulence dissipation rate for each bubble

`ububbm1(mgblb),vbubbm1(mgblb),wbubbm1(mgblb)`
velocity components for each bubble, previous time step

`ububbm2(mgblb),vbubbm2(mgblb),wbubbm2(mgblb)`
velocity components for each bubble, one before previous time step

`memoryi(mgblb),memoryj(mgblb),memoryk(mgblb)`
grid cell where each bubble was found during last search

`mconti(mgblb),id(mgblb)`
tags specifying new or old bubble and bubble identification of origin (point of injection)

`xcen(mgblb),ycen(mgblb),zcen(mgblb)`
position of each bubble

xcenm1(mgblb),ycenm1(mgblb),zcenm1(mgblb)
position of each bubble, previous time step

xcenm2(mgblb),ycenm2(mgblb),zcenm2(mgblb)
position of each bubble, one before previous time step

umean,vmean,wmean,pmean,xkmean,epsmean
velocity, pressure and turbulence dissipation energy sensed by bubble

umean1(mgblb),vmean1(mgblb),wmean1(mgblb),epsmean1(mgblb)
velocity, pressure and turbulence dissipation energy sensed by bubble, previous time step

rpm,radpsec
revolutions per minute and radians per second of the runner

velscale,scale
bulk inlet velocity (m/s) and diameter of the inlet plane (m)

acura, ibbac
specifications for the quality of the grid, the coarser or more skewed the grid, the greater these values must be for successful execution

rad(mgblb)
radius of each bubble

fxbub(mgblb),fybub(mgblb),fzbub(mgblb)
three Cartesian directions components of force acting on the bubble

gmass(mgblb)
air mass of each bubble

deng(mgblb),denw,viscw
density of the air of each bubble, density of water, dynamic viscosity of water

consdo,conssat
DO concentration of forebay water, saturation concentration of water

cdcoef
drag coefficient of the bubble

nosmooth, smoothfact
number of times the smoothing operator will be applied, weighting of the smoothing influence

All variable names ending with ...sw correspond to intermediate bubble arrays used for temporary storage of properties and swapping.

All variable names ending with ...br correspond to intermediate bubble arrays created from bubble breakup and are used for temporary storage of bubble properties.

The data file CONTROL

The code data file CONTROL allows the user to specify all the necessary data to the code. The file is structured in a self explanatory line: every line of actual data is preceded by a comment line, describing the data line that follows. This data file and all of the programming performed is using the metric (SI) unit system. A typical sample of the CONTROL file is:

```
Speed of turbine (rpm)
112.5
File name for the grid
plot3d.grd
File name for the velocity
velocity
File name for the pressure
P
File name for the turbulent Kinetic energy
TKE
File name for the dissipation of the turbulent Kinetic energy
EPSILON
File name for the wall iblanking specs
plot3d.wall
Turbulence model used (0 for k-e, 1 for k-w)
1
Forebay water temperature (C) and DO saturation concentration (kg/m^3)
25  0.0462
Accuracy factor of the interpolation, wall definition accuracy
1.1 4
Forebay water density, water dynamic viscosity, DO conc. (kg/m^3)
998.2  0.001002, 0.001
Href HLoverHREF VSQexover2gHref Pvapor PrefProto sigmaP Zref    VsqDTE (dt)
7.78 0.02       0.142           2450   241800    0.812  0.1289  0.
Geometry scale, velocity scale
20. 1.714
Max No of time steps, injection step and bubble write step
4000  20  50
Number of initial locations of air injection
20
x,y,z coord.(CFD units!) for inj. ports, type of injection, m3/sec of air and
radius of opening (prototype units!)
   0.1    0.1  -.3 1  .05     .02
…
 -0.02   0.02 -.3 1  .05     .02
Smoothing passes and smoothing factor
6 .3
```

The quote "type of injection" at the last line of input control whether the corresponding injection point is stationary (0) or rotating with the runner (1).

The common block `com-bubble`

The same identical common block file is included in every subroutine. This way, it is very easy to change the dimensions defining the CFD solution and grid sizes as well as the number of bubbles the code can store. The parameters appearing in this file (along with their respective meaning) are:

`mi`	is the maximum -I- direction grid capacity
`mj`	is the maximum -j- direction grid capacity
`mk`	is the maximum -k- direction grid capacity
`mif`	is an inactive constant which must always be set to 1
`mjf`	is an inactive constant which must always be set to 1
`mgblb`	is the total number of bubbles the program can simulate

The common block file is:

```
      parameter (mi=31,mj=53,mk=42,mbl=31)
      parameter (mif=1,mjf=1,mgblb=100000)
      common/control1/main,itraj,mtraj,time,itrajcount
      common/control2/maxts,jumpts,iwrite,isteper
      common/inject1/initnum,xinlet(100),yinlet(100)
      common/inject2/zinlet(100),inlettype(100),radorif(100)
      common/geom1/ix(mbl),iy(mbl),iz(mbl),volini(mi,mj,mk,9,mbl)
      common/geom2/x(mi,mj,mk,mbl),y(mi,mj,mk,mbl),z(mi,mj,mk,mbl)
      common/geom3/iwall(mi,mj,mk,mbl),ibl(mi,mj,mk,mbl),iblock
      common/flow1/u(mi,mj,mk,mbl),v(mi,mj,mk,mbl),w(mi,mj,mk,mbl)
      common/flow2/p(mi,mj,mk,mbl),xk(mi,mj,mk,mbl)
      common/flow3/eps(mi,mj,mk,mbl),reynolds,bbreak,iturbul
      common/constants/pi,gi,dt,temp,runiv,pabs,tabs,xctrans,xlamda
      common/fgeom/xbub(mif,mjf),ybub(mif,mjf),zbub(mif,mjf),ixf,jxf
      common/fval1/pbubble(mgblb),xkbubble(mgblb)
      common/fval2/ububble(mgblb),vbubble(mgblb),wbubble(mgblb)
      common/fval2m1/ububbm1(mgblb),vbubbm1(mgblb),wbubbm1(mgblb)
      common/fval2m2/ububbm2(mgblb),vbubbm2(mgblb),wbubbm2(mgblb)
      common/fforce/fxbub(mgblb),fybub(mgblb),brtime(mgblb),
     +fzbub(mgblb),xmass(mgblb),gmass(mgblb),perbubble
      common/fprop/deng(mgblb),denw,viscw,consdo,conssat,cdcoef
      common/stats/xkkilled,colkilled,coakilled
      common/che/iput(mif,mjf),jput(mif,mjf),kput(mif,mjf),
     +iblput(mif,mjf)
      common/helper/iputlast,jputlast,kputlast,acura,ibbac
      common/means/umean,vmean,wmean,pmean,xkmean,epsmean
      common/operat/rpm,radpsec,velscale,scale
      common/surrstore/umean1(mgblb),vmean1(mgblb),wmean1(mgblb),
     +epsmean1(mgblb)
      common/draft/Href,HLoverHREF,VSQexover2gHref,
     +Pvapor,PrefProto,sigmaP,Zref,VsqDTE
      common/direx/
     +xdir1(mi,mj,mk,mbl),xdir2(mi,mj,mk,mbl),xdir3(mi,mj,mk,mbl),
     +ydir1(mi,mj,mk,mbl),ydir2(mi,mj,mk,mbl),ydir3(mi,mj,mk,mbl),
     +zdir1(mi,mj,mk,mbl),zdir2(mi,mj,mk,mbl),zdir3(mi,mj,mk,mbl)
      common/bubs1/ ibub,ibubble,rad(mgblb),xcen(mgblb)
      common/bubs2/ ycen(mgblb),zcen(mgblb),cfsair(mgblb)
      common/bubs2m1/xcenm1(mgblb),ycenm1(mgblb),zcenm1(mgblb)
      common/bubs3m1/xcenm2(mgblb),ycenm2(mgblb),zcenm2(mgblb)
```

```
      common/bubs3/ memoryi(mgblb),memoryj(mgblb),ilomem(mgblb)
      common/bubs4/ memoryk(mgblb),mconti(mgblb),id(mgblb)
      common/swap/memoryisw(mgblb),memoryjsw(mgblb),brtimesw(mgblb),
     +memoryksw(mgblb),radsw(mgblb),xcensw(mgblb),ycensw(mgblb),
     +zcensw(mgblb),mcontisw(mgblb),idsw(mgblb),ububbm1sw(mgblb),
     +wbubbm1sw(mgblb),ububbm2sw(mgblb),vbubbm2sw(mgblb),
     +dengsw(mgblb),xmasssw(mgblb),gmasssw(mgblb),vbubbm1sw(mgblb),
     +wbubbm2sw(mgblb),xcenm1sw(mgblb),ycenm1sw(mgblb),
     +zcenm1sw(mgblb),xcenm2sw(mgblb),ycenm2sw(mgblb),
     +zcenm2sw(mgblb),ububblesw(mgblb),vbubblesw(mgblb),wbubblesw(mgblb)
      common/breakup/gmassbr(mgblb),
     +xcenbr(mgblb),
     +ycenbr(mgblb),
     +zcenbr(mgblb),
     +xcenm1br(mgblb),
     +ycenm1br(mgblb),
     +zcenm1br(mgblb),
     +xcenm2br(mgblb),
     +ycenm2br(mgblb),
     +zcenm2br(mgblb),
     +radbr(mgblb),
     +mcontibr(mgblb),
     +idbr(mgblb),
     +ilomemibr(mgblb),
     +memoryibr(mgblb),
     +memoryjbr(mgblb),
     +memorykbr(mgblb),
     +ububblebr(mgblb),
     +vbubblebr(mgblb),
     +wbubblebr(mgblb),
     +ububbm1br(mgblb),
     +vbubbm1br(mgblb),
     +wbubbm1br(mgblb),
     +ububbm2br(mgblb),
     +vbubbm2br(mgblb),
     +wbubbm2br(mgblb),
     +dengbr(mgblb)
      common/files/grfile,velfile,presfile,tkefile,epsfile,wallfile
      character*20 grfile,velfile,presfile,tkefile,epsfile,wallfile
      common/smoothing/ nosmooth, smoothfact,xsource(mi,mj,mk,mbl),
     +ysource(mi,mj,mk,mbl),zsource(mi,mj,mk,mbl),
     +xsourcesm(mi,mj,mk,mbl),
     +ysourcesm(mi,mj,mk,mbl),zsourcesm(mi,mj,mk,mbl)
```

APPENDIX B: The code `bubble.f`

A listing of the actual code follows. The code is thoroughly commented, standard Fortran has been used throughout and should be very easily comprehensible to an average Fortran programmer.

```
c ********************************************************
c
      program Virtual Bubble
c
c This program computes the trajectories of spherically shaped bubbles
c in draft tubes and estimates the DO exchange from the bubbles to the
c water.
c A lot of the coding in this program was originally oriented towards
c the tracking of a 3D, arbitrarily shaped body in a multiblock CFD
c solution domain. Most of the multi-block-related code has been
c cleaned, however a small part concerning the body surface coding
c is still here. This part is inactive, unusable and does not affect the
c execution speed.
c
c***************************************
c Version 1.2
c December 1999,
c***************************************
c
      include'com-bubble'
      character*1 zzz
c
c File 'CONTROL' contains the basic data necessary for each run. It is
c self-explanatory, since it is formatted in a way that allows one line
c of data to be preceeded by on line of description of this data.
c
      write(*,*)
      write(*,*) 'Reading control file'
c
c
      open(1,file='CONTROL')
  100 format(80a1)
c
c
      read(1,100) zzz
      read(1,*) rpm
      radpsec=(rpm*2.*3.14157)/60.
      read(1,100) zzz
      read(1,5) grfile
      read(1,100) zzz
      read(1,5) velfile
      read(1,100) zzz
      read(1,5) presfile
      read(1,100) zzz
      read(1,5) tkefile
      read(1,100) zzz
      read(1,5) epsfile
      read(1,100) zzz
      read(1,5) wallfile
  5   format(a20)
      read(1,100) zzz
```

```
      read(1,*) iturbul
      read(1,100) zzz
      read(1,*) temp,conssat
      read(1,100) zzz
      read(1,*) acura,ibbac
      read(1,100) zzz
      read(1,*) denw,viscw,consdo
c read for draft tube
      read(1,100) zzz
      read(1,*)Href,HLoverHREF,VSQexover2gHref,
     +Pvapor,PrefProto,sigmaP,Zref,VsqDTE
c
      read(1,100) zzz
      read(1,*) scale,velscale
      read(1,100) zzz
      read(1,*) maxts,jumpts,iwrite
      read(1,100) zzz
      read(1,*) initnum
      read(1,100) zzz
      do i=1,initnum
      read(1,*) xinlet(i),yinlet(i),zinlet(i),
     +inlettype(i),cfsair(i),radorif(i)
c
c bubble injection points are scaled to fit full scale geometry
      xinlet(i)=xinlet(i)*scale
      yinlet(i)=yinlet(i)*scale
      zinlet(i)=zinlet(i)*scale
c
      enddo
      read(1,100) zzz
      read(1,*) nosmooth, smoothfact

      close(1)
c
c
c Bubble tracking
c
c This subroutine computes the air/water+air flowrate xlamda
c
      call complamda
c
c Subroutine cnstnts gives values to all the hardcoded constants of the
c code, like acceleration of gravity etc.
c
      call cnstnts
c
c Subroutine readfield read the grid geometry (x,y,z) and the
c solution flowfield on that geometry (p,u,v,w,k,e)
c
      call readfield
c
c Subroutine dimension scales the data to full scale quantities
c
      call dimensions
c
c Subroutine geom pre-computes grid lines directions to enhance
c computational efficiency
c
      call geom
c
c In order to speed up the search algorithm, the main tetrahedron
```

```
c volumes are precomputed
c
      call prepare
c
c In order to enhance the spead of the search algorithm and the a
c accuracy of the integration, an "optimum" is precomputed
c
      call compdt
c
c Subroutine traject computes the trajectories of the bubbles
c and performs all necessary computations for the DO transfer
c estimation
c
      call traject
c
c Subroutine sources computes momentum source terms from the drag
c bubble put on the flow, to be used with a RANS solver
c
      call sources
c
c
      stop
      end
c
c ********************************************************
c

      subroutine cnstnts
c
c This subroutine specifies the values of all the important constants for
c the code. The Metric Unit System is used for all dimensional constants
c
      include'com-bubble'
c Pi
      pi=3.14159265359
c Acceleration of gravity
      gi=9.81
c Universal gas constant
      runiv=286.9
c Absolute pressure
      pabs=1.013E5
c Absolute temperature
      tabs=273.15
c reynolds number of the flow (needed for Kl formula)
      reynolds=velscale*scale*denw/viscw
      write(*,*) 'Flow Reynolds number is:', reynolds
c front part of air transfer formula
      xctrans=(8.33E-5)*(reynolds**(0.363))*xlamda**(-0.225)
c surface tension (sigma) of water
      sigma=0.00734
c critical bubble weber number
      wecr=1.1
c bubble breakup criterion term
      bbreak=((wecr/2.)**(0.6))*(sigma**(0.6))/((denw*denw)**(0.2))
c
c
c
      return
      end
c
c ********************************************************
```

```
c
      subroutine readfield
c
c
c This subroutine reads the grid, velocity, pressure and
c turbulence multiblock solutions as they are computed by
c CFD solver
      include'com-bubble'
      character*1, adum
c
c
c read the grid blocks
      open(10,file=grfile,form='unformatted')
      read(10) iblock
      write(*,*) iblock
      read(10) (ix(ilo),iy(ilo),iz(ilo),ilo=1,iblock)
      do 101 ilo=1,iblock
      write(*,*) ix(ilo),iy(ilo),iz(ilo)
      read(10) (((x(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo)),
     +       (((y(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo)),
     +       (((z(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo)),
     +       (((ibl(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
 101  continue
      close(10)
      write(*,*) 'Grid o.k.'
c
c read the velocity blocks
      open(10,file=velfile,form='unformatted')
      read(10) iblock
      read(10) (ix(ilo),iy(ilo),iz(ilo),idum,ilo=1,iblock)
      do 102 ilo=1,iblock
      read(10) (((u(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo)),
     +       (((v(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo)),
     +       (((w(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
 102  continue
      close(10)
      write(*,*) 'Velocity o.k.'
c
c read the pressure blocks
      open(10,file=presfile,form='unformatted')
      read(10) iblock
      read(10) (ix(ilo),iy(ilo),iz(ilo),idum,ilo=1,iblock)
      do 103 ilo=1,iblock
      read(10) (((p(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
 103  continue
      close(10)
      write(*,*) 'Pressure o.k.'
c
c read the turbulence kinetic energy blocks
      open(10,file=tkefile,form='unformatted')
      read(10) iblock
      read(10) (ix(ilo),iy(ilo),iz(ilo),idum,ilo=1,iblock)
      do 104 ilo=1,iblock
      read(10)(((xk(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
 104  continue
      close(10)
      write(*,*) 'Turbulence data 1 o.k.'
c
c read the dissipation of turbulence kinetic energy blocks
      open(10,file=epsfile,form='unformatted')
      read(10) iblock
```

```
      read(10) (ix(ilo),iy(ilo),iz(ilo),idum,ilo=1,iblock)
      do 105 ilo=1,iblock
      read(10)(((eps(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
 105  continue
      close(10)
      write(*,*) 'Turbulence data 2 o.k.'

c Read the Wall information
      open(10,file=wallfile,form='formatted')
      read(10,22) adum
      do ilo=1,iblock
      do i=1,ix(ilo)
      do j=1,iy(ilo)
      do k=1,iz(ilo)
      read(10,*) idum1,idum2,idum3,idum4,dumwall
      dumwall=abs(dumwall)
      iwall(idum2,idum3,idum4,idum1)=1
      if(dumwall.gt.10.e-12) then
      iwall(idum2,idum3,idum4,idum1)=0
      endif
      enddo
      enddo
      enddo
      enddo
 22   format(a1)
      close(10)
      write(*,*) 'Wall data o.k.'
c

      open(1,file='XYZ.dat')
      write(1,*) 'TITLE = "COORDS"'
      write(1,*) 'VARIABLES= X,Y,Z'
      do ilo=1,iblock
      write(1,*)
     +'ZONE T="1",I=',ix(ilo),',J=',iy(ilo),',K=',iz(ilo),' F=block'
      write(1,*) (((x(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((y(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((z(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      enddo
      close(1)
c
      open(1,file='EVERYTHING.dat')
      write(1,*) 'TITLE = "Everything"'
      write(1,*) 'VARIABLES= X,Y,Z,P,U,V,W,k,eps,iwall'
      do ilo=1,iblock
      write(1,*)
     +'ZONE T="1",I=',ix(ilo),',J=',iy(ilo),',K=',iz(ilo),' F=block'
      write(1,*) (((x(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((y(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((z(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((p(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((u(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((v(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((w(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((xk(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((eps(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((iwall(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      enddo
      close(1)
```

```
c
      return
      end
c
c ********************************************************
c
      subroutine initpos
c
c This subroutine specifies the initial position of
c the bubbles, i.e. their points of entry in the draft tube. If the
c unsteady option is activated for an inlet boundary, this boundary is
c rotated, to resemble the rotation of the runner.
c
      include'com-bubble'
c
      do 1 i=1,initnum
      if (inlettype(i).eq.0) then
      xcen(i)=xinlet(i)
      ycen(i)=yinlet(i)
      zcen(i)=zinlet(i)
      endif
      if (inlettype(i).eq.1) then
      zcen(i)=zinlet(i)
      xx=xinlet(i)*cos(time*radpsec)+yinlet(i)*sin(time*radpsec)
      yy=yinlet(i)*cos(time*radpsec)-xinlet(i)*sin(time*radpsec)
      xcen(i)=xx
      ycen(i)=yy
      endif
      id(i)=i
      mconti(i)=0
c initialize breakup delay (something very big)
      brtime(i)=10000000.
    1 continue
c
      return
      end
c
c ********************************************************
c
      subroutine traject
c
c This subroutine integrates the equations of motion for the
c bubble in the draft-tube and propagates the bubble.
c
      include'com-bubble'
      character*15 ffile1
      character*17 ffile2
      dimension nobubbs(100000)
c
      time=0.
      isteper=0
      ibubble=0
c
c Main loop identifier (back here whenever new bubbles are injected)
c
 1073 continue
c
c Initialize statistics
c
      xkkilled=0
      colkilled=0
```

```
      coakilled=0
c
c
c Inject new bubbles
c     initial position
      call initpos
c
      ibubble=ibubble+initnum
c
c Secondary loop identifier (back here every time step)
 3454 continue
c
c counter and time step increment
      isteper=isteper+1
      time=time+dt
      write(*,6699) isteper,time,ibubble
 6699 format('Step no: ',i7,' at time ',e12.5,' with ',i6,' bubbles')
c
c loop scaning all bubbles per time step
      do 9999 ibub=1,ibubble
c
c prepape data for locate
      xbub(1,1)=xcen(ibub)
      ybub(1,1)=ycen(ibub)
      zbub(1,1)=zcen(ibub)
      iconti=mconti(ibub)
c
c find position of bubble
      call locate(iconti)
c compute ambient flow field
      call ambient
c determine properties of bubble
      call physics
c compute forces exerted on bubble
      call forces
c propagate bubble to its new location
      call march
c mark bubble as "old"
      mconti(ibub)=1
c
c end of "every bubble" loop
c
 9999 continue
c
c Bubble passes air to the water
c
      do 4634 i=1,ibubble
      uxrel=ububble(i)-umean1(i)
      vyrel=vbubble(i)-vmean1(i)
      wzrel=wbubble(i)-wmean1(i)
      velrel=sqrt(uxrel**2+vyrel**2+wzrel**2)
      surface=4.*pi*rad(i)**2
      defic=conssat-consdo
      trans=dt*xctrans*velrel*surface*defic
      if((gmass(i)-trans).gt.(1.e-20)) then
      gmass(i)=gmass(i)-trans
c XXX put trans in proper sum
      else
      id(i)=0
c XXXX Pass all air to water
      write(*,*) 'bubble',i,' passed all air to water'
```

```
      endif
 4634 continue

c
c The following evaluation for the fate of the bubble takes
c place at the new location. This implies that we accept that
c none of the following criteria are satisfied at the injection
c location.
c
c Evaluate possible bubble breakup
c Bubble Breakup occurs when Hasketh criterion is
c satisfied. Bubble breakup is binary.
c
      do 8226 i=1,ibubble
      if (id(i).eq.0) goto 8226
c Critical radius per Hesketh
      dcrhesk=bbreak*(epsmean1(i)**(-0.4))/(deng(i)**(0.2))
c     write(*,*) 'hesk',2.*rad(i), dcrhesk,i
      if ((2.*rad(i)).gt.dcrhesk) then
      brtime(i)=amin1(brtime(i),(time+4.5))
      endif
 8226 continue
c
      ibroken=0
      do 8227 i=1,ibubble
      if(brtime(i).le.time) then
      ibroken=ibroken+1
c     write(*,*) 'bubble broke!'
c equi-distribution of mass
      gmass(i)=gmass(i)/2.
c new radius
      rad(i)= rad(i)/1.2599
      brtime(i)=10000000.
c call subroutine to arrange new matrix
      call create_br(gmass(i),rad(i),ibroken,i)
      endif
 8227 continue
c
c attach the bubbles from breakup to the main bubble arrays
c
      if(ibroken.ne.0) then
      call newbubs(ibroken)
      do i=1,ibubble
c     write(*,*) 'olaxcen',i,xcen(i)
      enddo
      endif
c
c
      do 8877 i=1,ibubble
c
c Evaluate possible bubble coalescence
c Bubble coalescence occurs when the distance between 2
c bubble centers is smaller that 1.2 times the sum of their radii
c 1.2 is a factor that accounts for local pressure reduction due
c to flow acceleration between bubbles, and deviation from
c perfect-spherical shape
c
      do 8874 j=1,ibubble
      if (id(i).eq.0) goto 8874
      if (id(j).eq.0) goto 8874
      if (i.eq.j) goto 8874
```

```
      distbb=sqrt(((xcen(i)-xcen(j))**2)+
     +((ycen(i)-ycen(j))**2)+((zcen(i)-zcen(j))**2))
      totrad=rad(i)+rad(j)
      if((1.2*totrad).gt.distbb) then
c     write(*,*) i,j,id(i),id(j),distbb,1.2*totrad,' coal'
      coakilled=coakilled+1
      id(i)=0
      gmass(j)=gmass(j)+gmass(i)
      brtime(j)=amin1(brtime(i),brtime(j))
      goto 8877
      endif
 8874 continue
 8877 continue
c
c If a bubble gets very close to the solid wall, it is bound
c to create a pocket of air there. Take such bubbles out of
c circulation.
c
      do 8821 i=1,ibubble
      itrajend=0
      call walls(itrajend,memoryi(i),memoryj(i),memoryk(i),ilomem(i))
      if(itrajend.eq.1) id(i)=0
 8821 continue
c
c clear the original bubble arrays from inexistant
c bubbles
      call swapthem
c
c Output
c
c
c These are the full results files (To be used with a post-processor).
      if(mod(isteper,iwrite).eq.0) then
      write(ffile2,300) 'RESULTS-',isteper,'.plt'
  300 format(a8,i5.5,a4)
      open(18,file=ffile2)
      do i=1,ibubble
      write(18,19)time,isteper,id(i),
     +xcen(i)/scale,ycen(i)/scale,zcen(i)/scale,
     +rad(i),gmass(i)
      enddo
      close(18)
      endif
c
c these is the simple Animation Files (to be used with the XYZ.plt geometry
file)
c
      if(mod(isteper,iwrite).eq.0) then
      write(ffile1,400) 'ANIM1-',isteper,'.plt'
  400 format(a6,i5.5,a4)

      open(18,file=ffile1)
       write(18,*) 'TITLE = "Just_COORD"'
       write(18,*) 'VARIABLES= X,Y,Z'
      do i=1,ibubble
      write(18,*)xcen(i)/scale,ycen(i)/scale,zcen(i)/scale
c     write(18,19)time,isteper,id(i),xcen(i),ycen(i),
c    +zcen(i),rad(i),gmass(i)
      enddo
      close(18)
      endif
```

```
c end (or not) the simulation
c
c     check for maximum number of steps
c
      if (isteper.eq.maxts) goto 1111
c
c Check for relatively stationary number of bubbles
c
      if (isteper.gt.1000) then
c
      itermin=0
      do iijj=isteper-3*jumpts+1, isteper-1
      ifiveperc=int(0.02*float(nobubbs(isteper-1)))
c     ibubbound=max0(ifiveperc,initnum)
      ibubbound=ifiveperc
      iupper=nobubbs(isteper-1)+ibubbound
      ilower=nobubbs(isteper-1)-ibubbound
      write(*,*) iupper,ilower,nobubbs(iijj),itermin
      if((ilower.le.nobubbs(iijj)).and.
     +   (nobubbs(iijj).le.iupper)) then
      itermin=itermin+1
      endif
      enddo
      if (itermin.gt.2*jumpts) goto 1111
c
      endif
c
c
c
c  Set the "keep track of bubbles in time step" array
c
      nobubbs(isteper)=ibubble
c inject (or not) new bubbles
      if (mod(isteper,jumpts).ne.0) goto 3454
c if injection is decided, shift old bubbles by
c initnum places in their arrays, to make space
      call shifter

      goto 1073
 1111 continue
c
c
      return
      end
c
c ***********************************************************
c
      subroutine locate(iconti)
c
c This subroutine finds in what cell of the computational
c mesh, the center of the bubble is.
c Note: This routine and the subroutines/functions called from
c this one, are the core of this program. Do not change anything
c unless you are absolutely sure you know what you are doing.
c
      include'com-bubble'
c
c These arrays hold the vertices of each cell (pos. 2-9)
c and the bubble center (pos. 1), temporarily for each locate scan
c
```

```
      dimension xt(9),yt(9),zt(9)
c
c Generalized 3D lattice locator matrices
c
c small lattice
c
      dimension ip1(6),ip2(6),ip3(6),ip4(6)
      data ip1
     +/3,7,7,4,2,3/
      data ip2
     +/4,8,9,5,3,5/
      data ip3
     +/6,9,6,6,5,2/
      data ip4
     +/7,4,4,9,6,6/
c
c
c see discusion on time step
c
      ispan=1
      if=1
      jf=1
c
c define search subdomain
      if(iconti.eq.1) then
c
      ilo=ilomem(ibub)
c
      istart=memoryi(ibub)-ispan
      jstart=memoryj(ibub)-ispan
      kstart=memoryk(ibub)-ispan
      iend=memoryi(ibub)+ispan
      jend=memoryj(ibub)+ispan
      kend=memoryk(ibub)+ispan
c
      if (istart.lt.1) istart=1
      if (jstart.lt.1) jstart=1
      if (kstart.lt.1) kstart=1
      if (iend.gt.ix(ilo)-1) iend=ix(ilo)-1
      if (jend.gt.iy(ilo)-1) jend=iy(ilo)-1
      if (kend.gt.iz(ilo)-1) kend=iz(ilo)-1
c
c search subdomain
      do 20 i=istart,iend
      do 20 j=jstart,jend
      do 20 k=kstart,kend
c
      xt(1)=xbub(if,jf)
      yt(1)=ybub(if,jf)
      zt(1)=zbub(if,jf)
      xt(2)=x(i,j,k,ilo)
      yt(2)=y(i,j,k,ilo)
      zt(2)=z(i,j,k,ilo)
      xt(3)=x(i,j,k+1,ilo)
      yt(3)=y(i,j,k+1,ilo)
      zt(3)=z(i,j,k+1,ilo)
      xt(4)=x(i,j+1,k+1,ilo)
      yt(4)=y(i,j+1,k+1,ilo)
      zt(4)=z(i,j+1,k,ilo)
      xt(5)=x(i,j+1,k,ilo)
      yt(5)=y(i,j+1,k,ilo)
```

```
      zt(5)=z(i,j+1,k,ilo)
      xt(6)=x(i+1,j,k,ilo)
      yt(6)=y(i+1,j,k,ilo)
      zt(6)=z(i+1,j,k,ilo)
      xt(7)=x(i+1,j,k+1,ilo)
      yt(7)=y(i+1,j,k+1,ilo)
      zt(7)=z(i+1,j,k+1,ilo)
      xt(8)=x(i+1,j+1,k+1,ilo)
      yt(8)=y(i+1,j+1,k+1,ilo)
      zt(8)=z(i+1,j+1,k+1,ilo)
      xt(9)=x(i+1,j+1,k,ilo)
      yt(9)=y(i+1,j+1,k,ilo)
      zt(9)=z(i+1,j+1,k,ilo)
c
      do 5 ilat=1,6
      iii=ip1(ilat)
      jjj=ip2(ilat)
      kkk=ip3(ilat)
      lll=ip4(ilat)
c
c
      volinaki=volini(i,j,k,ilat,ilo)
      idecis=
     +icheck(xt(1),xt(iii),xt(jjj),xt(kkk),xt(lll)
     +,      yt(1),yt(iii),yt(jjj),yt(kkk),yt(lll)
     +,      zt(1),zt(iii),zt(jjj),zt(kkk),zt(lll),acura,
     +volinaki)
c
      if (idecis.eq.1) then
c
      iblput(if,jf)=ilo
      iput(if,jf)=i
      jput(if,jf)=j
      kput(if,jf)=k
c
      ilomem(ibub)=ilo
      memoryi(ibub)=i
      memoryj(ibub)=j
      memoryk(ibub)=k
c     write(*,*) iput(if,jf),jput(if,jf),kput(if,jf)
      goto 10
      endif
   5  continue
  20  continue
c
c  this closes the if statement for old bubbles
      endif
c
c This is for old bubbles that failed to be found immediately
c or for new bubbles
c
c search everything
c
      do 21 ilo=1,iblock
      do 21 i=1,ix(ilo)-1
      do 21 j=1,iy(ilo)-1
      do 21 k=1,iz(ilo)-1
c
      xt(1)=xbub(if,jf)
      yt(1)=ybub(if,jf)
      zt(1)=zbub(if,jf)
```

```
      xt(2)=x(i,j,k,ilo)
      yt(2)=y(i,j,k,ilo)
      zt(2)=z(i,j,k,ilo)
      xt(3)=x(i,j,k+1,ilo)
      yt(3)=y(i,j,k+1,ilo)
      zt(3)=z(i,j,k+1,ilo)
      xt(4)=x(i,j+1,k+1,ilo)
      yt(4)=y(i,j+1,k+1,ilo)
      zt(4)=z(i,j+1,k,ilo)
      xt(5)=x(i,j+1,k,ilo)
      yt(5)=y(i,j+1,k,ilo)
      zt(5)=z(i,j+1,k,ilo)
      xt(6)=x(i+1,j,k,ilo)
      yt(6)=y(i+1,j,k,ilo)
      zt(6)=z(i+1,j,k,ilo)
      xt(7)=x(i+1,j,k+1,ilo)
      yt(7)=y(i+1,j,k+1,ilo)
      zt(7)=z(i+1,j,k+1,ilo)
      xt(8)=x(i+1,j+1,k+1,ilo)
      yt(8)=y(i+1,j+1,k+1,ilo)
      zt(8)=z(i+1,j+1,k+1,ilo)
      xt(9)=x(i+1,j+1,k,ilo)
      yt(9)=y(i+1,j+1,k,ilo)
      zt(9)=z(i+1,j+1,k,ilo)
c
      do 51 ilat=1,6
      iii=ip1(ilat)
      jjj=ip2(ilat)
      kkk=ip3(ilat)
      lll=ip4(ilat)
c
c
      volinaki=volini(i,j,k,ilat,ilo)
      idecis=
     +icheck(xt(1),xt(iii),xt(jjj),xt(kkk),xt(lll)
     +,      yt(1),yt(iii),yt(jjj),yt(kkk),yt(lll)
     +,      zt(1),zt(iii),zt(jjj),zt(kkk),zt(lll),acura,
     +volinaki)
c
      if (idecis.eq.1) then
c
      iblput(if,jf)=ilo
      iput(if,jf)=i
      jput(if,jf)=j
      kput(if,jf)=k
c
      ilomem(ibub)=ilo
      memoryi(ibub)=i
      memoryj(ibub)=j
      memoryk(ibub)=k
      goto 10
      endif
   51 continue
   21 continue
c
      id(ibub)=0
c     write(*,*) 'Finally not found! I,J,K', memoryi(ibub),
c    +memoryj(ibub),memoryk(ibub)
c     write(*,*) 'uvwmean', umean,vmean,wmean
c     write(*,*) 'uvwbub', ububble(ibub),vbubble(ibub),wbubble(ibub)
c     write(*,*) 'xyzcen', xcen(ibub),ycen(ibub),zcen(ibub)
```

```
c     write(*,*) 'cell'
c     write(*,*)  x(memoryi(ibub),memoryj(ibub),memoryk(ibub)),
c    +y(memoryi(ibub),memoryj(ibub),memoryk(ibub)),
c    +z(memoryi(ibub),memoryj(ibub),memoryk(ibub))
c     write(*,*)  x(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub)),
c    +y(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub)),
c    +z(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub))
c     write(*,*)  x(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub)+1),
c    +y(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub)+1),
c    +z(memoryi(ibub),memoryj(ibub)+1,memoryk(ibub)+1)
c     write(*,*)  x(memoryi(ibub),memoryj(ibub),memoryk(ibub)+1),
c    +y(memoryi(ibub),memoryj(ibub),memoryk(ibub)+1),
c    +z(memoryi(ibub),memoryj(ibub),memoryk(ibub)+1)
c     write(*,*)  x(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub)),
c    +y(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub)),
c    +z(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub))
c     write(*,*)  x(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub)),
c    +y(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub)),
c    +z(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub))
c     write(*,*)  x(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub)+1),
c    +y(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub)+1),
c    +z(memoryi(ibub)+1,memoryj(ibub)+1,memoryk(ibub)+1)
c     write(*,*)  x(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub)+1),
c    +y(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub)+1),
c    +z(memoryi(ibub)+1,memoryj(ibub),memoryk(ibub)+1)
c
   10 continue
      return
      end
c
c ********************************************************
c
      integer function icheck(xf1,xijk,xi1jk,xij1k,xijk1,
     +yf1,yijk,yi1jk,yij1k,yijk1,zf1,zijk,zi1jk,zij1k,zijk1,acura,
     +volinaki)
c
c This function examines if the center of a bubble is in
c a particular cell of the grid
c
      icheck=0
c
c volume of main tetrahedron (this is precomputed)
c
      vol1f=volinaki
c
c subvolumes inside tetrahedron 1
c
c subvolume 1
c
      vol11=volu(xf1,xi1jk,xij1k,xijk1,
     +           yf1,yi1jk,yij1k,yijk1,
     +           zf1,zi1jk,zij1k,zijk1)
c
c subvolume 2
c
      vol12=volu(xf1,xijk,xij1k,xijk1,
     +           yf1,yijk,yij1k,yijk1,
     +           zf1,zijk,zij1k,zijk1)
c
c subvolume 3
c
```

```
      vol13=volu(xf1,xi1jk,xijk,xijk1,
     +           yf1,yi1jk,yijk,yijk1,
     +           zf1,zi1jk,zijk,zijk1)
c
c subvolume 4
c
      vol14=volu(xf1,xi1jk,xij1k,xijk,
     +           yf1,yi1jk,yij1k,yijk,
     +           zf1,zi1jk,zij1k,zijk)
c
      vol1c=vol11+vol12+vol13+vol14
      voldif=abs(vol1f-vol1c)/vol1f
      if (voldif.le.acura) then
      icheck=1
      endif
      return
      end
c
c *******************************************************
c
      real function volu(x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,z3,z4)
c this function computes the volume of a tetrahedron with
c vertices (x,y,z)_1,2,3,4
c
      dx1=x2-x1
      dx2=x3-x1
      dx3=x4-x1
      dy1=y2-y1
      dy2=y3-y1
      dy3=y4-y1
      dz1=z2-z1
      dz2=z3-z1
      dz3=z4-z1
      volu =abs(dx1*dy2*dz3+dy1*dz2*dx3+dz1*dx2*dy3-
     -                  dy1*dx2*dz3-dx1*dz2*dy3-dz1*dy2*dx3)
c
c Note: The exact formula of the volume of a tetrahedron requires
c multiplication by 1/6, a term that can and is omited (since
c only comparisons of volumes take place) for the sake of performance.
c
      return
      end
c
c *******************************************************
c
      subroutine ambient
c
c This subroutine determines the local conditions the bubble is
c sensing in its current location.
c
      include'com-bubble'
      ilo=iblput(1,1)
      d1=
     +sqrt((xbub(1,1)-x(iput(1,1),jput(1,1),kput(1,1),ilo))**2
     ++(xbub(1,1)-x(iput(1,1),jput(1,1),kput(1,1),ilo))**2
     ++(ybub(1,1)-y(iput(1,1),jput(1,1),kput(1,1),ilo))**2
     ++(zbub(1,1)-z(iput(1,1),jput(1,1),kput(1,1),ilo))**2)
      d2=
     +sqrt((xbub(1,1)-x(iput(1,1),jput(1,1),kput(1,1)+1,ilo))**2
     ++(ybub(1,1)-y(iput(1,1),jput(1,1),kput(1,1)+1,ilo))**2
     ++(zbub(1,1)-z(iput(1,1),jput(1,1),kput(1,1)+1,ilo))**2)
```

```
 d3=
+sqrt((xbub(1,1)-x(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo))**2
++(ybub(1,1)-y(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo))**2
++(zbub(1,1)-z(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo))**2)
 d4=
+sqrt((xbub(1,1)-x(iput(1,1),jput(1,1)+1,kput(1,1),ilo))**2
++(ybub(1,1)-y(iput(1,1),jput(1,1)+1,kput(1,1),ilo))**2
++(zbub(1,1)-z(iput(1,1),jput(1,1)+1,kput(1,1),ilo))**2)
 d5=
+sqrt((xbub(1,1)-x(iput(1,1)+1,jput(1,1),kput(1,1),ilo))**2
++(ybub(1,1)-y(iput(1,1)+1,jput(1,1),kput(1,1),ilo))**2
++(zbub(1,1)-z(iput(1,1)+1,jput(1,1),kput(1,1),ilo))**2)
 d6=
+sqrt((xbub(1,1)-x(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo))**2
++(ybub(1,1)-y(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo))**2
++(zbub(1,1)-z(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo))**2)
 d7=
+sqrt((xbub(1,1)-x(iput(1,1)+1,jput(1,1)+1,
+kput(1,1)+1,ilo))**2
++(ybub(1,1)-y(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo))**2
++(zbub(1,1)-z(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo))**2)
 d8=
+sqrt((xbub(1,1)-x(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))**2
++(ybub(1,1)-y(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))**2
++(zbub(1,1)-z(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))**2)
 d1=d1**(-3.5)
 d2=d2**(-3.5)
 d3=d3**(-3.5)
 d4=d4**(-3.5)
 d5=d5**(-3.5)
 d6=d6**(-3.5)
 d7=d7**(-3.5)
 d8=d8**(-3.5)
 dtot=d1+d2+d3+d4+d5+d6+d7+d8
 umean=(d1*u(iput(1,1),jput(1,1),kput(1,1),ilo)
++d2*u(iput(1,1),jput(1,1),kput(1,1)+1,ilo)
++d3*u(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo)
++d4*u(iput(1,1),jput(1,1)+1,kput(1,1),ilo)
++d5*u(iput(1,1)+1,jput(1,1),kput(1,1),ilo)
++d6*u(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo)
++d7*u(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo)
++d8*u(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))/dtot
 vmean=(d1*v(iput(1,1),jput(1,1),kput(1,1),ilo)
++d2*v(iput(1,1),jput(1,1),kput(1,1)+1,ilo)
++d3*v(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo)
++d4*v(iput(1,1),jput(1,1)+1,kput(1,1),ilo)
++d5*v(iput(1,1)+1,jput(1,1),kput(1,1),ilo)
++d6*v(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo)
++d7*v(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo)
++d8*v(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))/dtot
 wmean=(d1*w(iput(1,1),jput(1,1),kput(1,1),ilo)
++d2*w(iput(1,1),jput(1,1),kput(1,1)+1,ilo)
++d3*w(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo)
++d4*w(iput(1,1),jput(1,1)+1,kput(1,1),ilo)
++d5*w(iput(1,1)+1,jput(1,1),kput(1,1),ilo)
++d6*w(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo)
++d7*w(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo)
++d8*w(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))/dtot
 pmean=(d1*p(iput(1,1),jput(1,1),kput(1,1),ilo)
++d2*p(iput(1,1),jput(1,1),kput(1,1)+1,ilo)
++d3*p(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo)
```

```
     ++d4*p(iput(1,1),jput(1,1)+1,kput(1,1),ilo)
     ++d5*p(iput(1,1)+1,jput(1,1),kput(1,1),ilo)
     ++d6*p(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo)
     ++d7*p(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo)
     ++d8*p(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))/dtot
      epsmean=(d1*eps(iput(1,1),jput(1,1),kput(1,1),ilo)
     ++d2*eps(iput(1,1),jput(1,1),kput(1,1)+1,ilo)
     ++d3*eps(iput(1,1),jput(1,1)+1,kput(1,1)+1,ilo)
     ++d4*eps(iput(1,1),jput(1,1)+1,kput(1,1),ilo)
     ++d5*eps(iput(1,1)+1,jput(1,1),kput(1,1),ilo)
     ++d6*eps(iput(1,1)+1,jput(1,1),kput(1,1)+1,ilo)
     ++d7*eps(iput(1,1)+1,jput(1,1)+1,kput(1,1)+1,ilo)
     ++d8*eps(iput(1,1)+1,jput(1,1)+1,kput(1,1),ilo))/dtot
      umean1(ibub)=umean
      vmean1(ibub)=vmean
      wmean1(ibub)=wmean
      epsmean1(ibub)=epsmean
  10  continue
      return
      end
c
c ********************************************************
c
      subroutine march
c
c This subroutine *advances* the bubble to the next location
c along its trajectory. "Advances" means time marching integration
c of all three equations of motion
c
      include'com-bubble'
c
c
c F = Me * du/dt  Integration
c
      ububble(ibub)=
     +(4.*ububbm1(ibub)-ububbm2(ibub)+
     +(2.*dt*fxbub(ibub)/xmass(ibub)))/3.
      vbubble(ibub)=
     +(4.*vbubbm1(ibub)-vbubbm2(ibub)+
     +(2.*dt*fybub(ibub)/xmass(ibub)))/3.
      wbubble(ibub)=
     +(4.*wbubbm1(ibub)-wbubbm2(ibub)+
     +(2.*dt*fzbub(ibub)/xmass(ibub)))/3.
c
c update values of u,v,w m1 & m2  (leaves m1 = current)
c
      ububbm2(ibub)=ububbm1(ibub)
      vbubbm2(ibub)=vbubbm1(ibub)
      wbubbm2(ibub)=wbubbm1(ibub)
      ububbm1(ibub)=ububble(ibub)
      vbubbm1(ibub)=vbubble(ibub)
      wbubbm1(ibub)=wbubble(ibub)

c
c U = dx/dt Integration
      xcen(ibub)=(2.*dt*ububble(ibub)+
     +4.*xcenm1(ibub)-xcenm2(ibub))/3.
      ycen(ibub)=(2.*dt*vbubble(ibub)+
     +4.*ycenm1(ibub)-ycenm2(ibub))/3.
      zcen(ibub)=(2.*dt*wbubble(ibub)+
     +4.*zcenm1(ibub)-zcenm2(ibub))/3.
```

```
c
c update values of x,y,z m1 & m2  (leaves m1 = current)
c
      xcenm2(ibub)=xcenm1(ibub)
      ycenm2(ibub)=ycenm1(ibub)
      zcenm2(ibub)=zcenm1(ibub)
      xcenm1(ibub)=xcen(ibub)
      ycenm1(ibub)=ycen(ibub)
      zcenm1(ibub)=zcen(ibub)

      return
      end
c
c ********************************************************
c
      subroutine prepare
c
c This subroutine precomputes main cell volumes for speed
c
      include'com-bubble'
c
c These arrays hold the vertices of each cell (pos. 2-9)
c and the bubble center, temporarily for each locate scan
c
      dimension xt(9),yt(9),zt(9)
c
c Generalized 3D lattice locator matrices
c
c small lattice
c
      dimension ip1(6),ip2(6),ip3(6),ip4(6)
      data ip1
     +/3,7,7,4,2,3/
      data ip2
     +/4,8,9,5,3,5/
      data ip3
     +/6,9,6,6,5,2/
      data ip4
     +/7,4,4,9,6,6/
c
      write(*,*) 'Preparing initial volumes'
c
      do 20 ilo=1,iblock
      do 20 i=1,ix(ilo)-1
      do 20 j=1,iy(ilo)-1
      do 20 k=1,iz(ilo)-1
c
c New multi searcher
c
      xt(1)=0.
      yt(1)=0.
      zt(1)=0.
      xt(2)=x(i,j,k,ilo)
      yt(2)=y(i,j,k,ilo)
      zt(2)=z(i,j,k,ilo)
      xt(3)=x(i,j,k+1,ilo)
      yt(3)=y(i,j,k+1,ilo)
      zt(3)=z(i,j,k+1,ilo)
      xt(4)=x(i,j+1,k+1,ilo)
      yt(4)=y(i,j+1,k+1,ilo)
      zt(4)=z(i,j+1,k,ilo)
```

```
      xt(5)=x(i,j+1,k,ilo)
      yt(5)=y(i,j+1,k,ilo)
      zt(5)=z(i,j+1,k,ilo)
      xt(6)=x(i+1,j,k,ilo)
      yt(6)=y(i+1,j,k,ilo)
      zt(6)=z(i+1,j,k,ilo)
      xt(7)=x(i+1,j,k+1,ilo)
      yt(7)=y(i+1,j,k+1,ilo)
      zt(7)=z(i+1,j,k+1,ilo)
      xt(8)=x(i+1,j+1,k+1,ilo)
      yt(8)=y(i+1,j+1,k+1,ilo)
      zt(8)=z(i+1,j+1,k+1,ilo)
      xt(9)=x(i+1,j+1,k,ilo)
      yt(9)=y(i+1,j+1,k,ilo)
      zt(9)=z(i+1,j+1,k,ilo)
c
      do 5 ilat=1,6
      iii=ip1(ilat)
      jjj=ip2(ilat)
      kkk=ip3(ilat)
      lll=ip4(ilat)
c
      call
     +inivol(xt(1),xt(iii),xt(jjj),xt(kkk),xt(lll)
     +,      yt(1),yt(iii),yt(jjj),yt(kkk),yt(lll)
     +,      zt(1),zt(iii),zt(jjj),zt(kkk),zt(lll),volare)
c
      volini(i,j,k,ilat,ilo)=volare
    5 continue
c
   20 continue
      write(*,*) 'Done preparing initial volumes'
      return
      end
c
c *******************************************************
c
      subroutine inivol(xf1,xijk,xi1jk,xij1k,xijk1,
     +yf1,yijk,yi1jk,yij1k,yijk1,zf1,zijk,zi1jk,zij1k,zijk1,
     +volare)
c
c This is the subroutine where the actual volumes are computed
c initially for speed.
c
c
c volume of main tetrahedron
c
      volare=volu(xijk,xi1jk,xij1k,xijk1,
     +           yijk,yi1jk,yij1k,yijk1,
     +           zijk,zi1jk,zij1k,zijk1)
      return
      end
c
c *******************************************************
c
      subroutine compdt
c
c Ensure that the time step for the integration is small enough
c to keep maximum step within 2 neighbouring cells. Beneficial
c for accuracy and mainly for speed, since only 9 cells need to
c be scanned.
```

```
c
      include'com-bubble'
c
      dt=10.e30
      do 34 ilo=1,iblock
      do 34 i=2,ix(ilo)-1
      do 34 j=2,iy(ilo)-1
      do 34 k=2,iz(ilo)-1
      xl1=sqrt(
     +((x(i+1,j,k,ilo)-x(i,j,k,ilo))**2)+
     +((y(i+1,j,k,ilo)-y(i,j,k,ilo))**2)+
     +((z(i+1,j,k,ilo)-z(i,j,k,ilo))**2))
      xl2=sqrt(
     +((x(i,j+1,k,ilo)-x(i,j,k,ilo))**2)+
     +((y(i,j+1,k,ilo)-y(i,j,k,ilo))**2)+
     +((z(i,j+1,k,ilo)-z(i,j,k,ilo))**2))
      xl3=sqrt(
     +((x(i,j,k+1,ilo)-x(i,j,k,ilo))**2)+
     +((y(i,j,k+1,ilo)-y(i,j,k,ilo))**2)+
     +((z(i,j,k+1,ilo)-z(i,j,k,ilo))**2))
      xlmin=amin1(xl1,xl2,xl3)
      if(xlmin.lt.0.00001) write(*,*) 'Problem with grid at ',i,j,k
      velmax=
     +amax1(abs(u(i,j,k,ilo)),abs(v(i,j,k,ilo)),abs(w(i,j,k,ilo)))
      dt=amin1(dt,(xlmin/velmax))
   34 continue
c If a lot of bubbles fail to be found, decrease this coefficient
      dt=4.*dt
      write(*,*) 'dt computed=',dt
c
      return
      end
c
c *******************************************************
c
      subroutine forces
      include'com-bubble'
c
c This subroutine adds up all the forces exerted on the
c bubble, along the three principle directions.
c
c bubble volume
      volbub=1.3333333333*pi*rad(ibub)*rad(ibub)*rad(ibub)
c bubble frontal area
      volfac=pi*rad(ibub)*rad(ibub)
c mass of gas
      rmass=deng(ibub)*volbub
c effective mass
      xmass(ibub)=rmass+0.5*volbub*denw
c viscous forces
      if(mconti(ibub).eq.1) then
      uxrel=ububble(ibub)-umean
      vyrel=vbubble(ibub)-vmean
      wzrel=wbubble(ibub)-wmean
      velrel=sqrt(uxrel**2+vyrel**2+wzrel**2)
      call cdreyn(velrel)
      dragforce=0.5*cdcoef*volfac*denw*velrel**2
      xvisc=-(dragforce/velrel)*uxrel
      yvisc=-(dragforce/velrel)*vyrel
      zvisc=-(dragforce/velrel)*wzrel
      else
```

```
c for newly injected isokinematic bubbles, save time
      xvisc=0.
      yvisc=0.
      zvisc=0.
      dragforce=0.
      endif
c pressure force
      xpres1=volfac*rad(ibub)*
     +(p(iput(1,1)+1,jput(1,1),kput(1,1),iblput(1,1))-
     +p(iput(1,1),jput(1,1),kput(1,1),iblput(1,1)))
      ypres1=volfac*rad(ibub)*
     +(p(iput(1,1),jput(1,1)+1,kput(1,1),iblput(1,1))-
     +p(iput(1,1),jput(1,1),kput(1,1),iblput(1,1)))
      zpres1=volfac*rad(ibub)*
     +(p(iput(1,1),jput(1,1),kput(1,1)+1,iblput(1,1))-
     +p(iput(1,1),jput(1,1),kput(1,1),iblput(1,1)))
      xpres=xpres1*xdir1(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +ypres1*ydir1(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +zpres1*zdir1(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))
      ypres=xpres1*xdir2(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +ypres1*ydir2(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +zpres1*zdir2(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))
      zpres=xpres1*xdir3(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +ypres1*ydir3(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))+
     +zpres1*zdir3(iput(1,1),jput(1,1),kput(1,1),iblput(1,1))
      xpres=-xpres
      ypres=-ypres
      zpres=-zpres
c buoyancy
      dendif=denw-deng(ibub)
      bforce=-dendif*volbub*gi
c total force
      fxbub(ibub)=xvisc+xpres+bforce
      fybub(ibub)=yvisc+ypres
      fzbub(ibub)=zvisc+zpres
      return
      end

c
c ***********************************************************
c
      subroutine physics
      include'com-bubble'
c
c This subroutine applies simple physical and kinematical
c laws to new and old bubbles
c
      if (mconti(ibub).eq.0) then
c Bubbles are supposed to be injected isokinematicaly.
c This section sets the velocity for this and previous
c time steps and the positions of previous time steps,
c for new bubbles.
      ububble(ibub)=umean
      vbubble(ibub)=vmean
      wbubble(ibub)=wmean
      ububbm1(ibub)=umean
      vbubbm1(ibub)=vmean
      wbubbm1(ibub)=wmean
      ububbm2(ibub)=umean
      vbubbm2(ibub)=vmean
      wbubbm2(ibub)=wmean
```

```
      xcenm1(ibub)=xcen(ibub)-umean*dt
      ycenm1(ibub)=ycen(ibub)-vmean*dt
      zcenm1(ibub)=zcen(ibub)-wmean*dt
      xcenm2(ibub)=xcenm1(ibub)-umean*dt
      ycenm2(ibub)=ycenm1(ibub)-vmean*dt
      zcenm2(ibub)=zcenm1(ibub)-wmean*dt
c
c Define new bubble size and mass
c
      deng(ibub)=(pmean+pabs)/(runiv*(temp+tabs))
      rad(ibub)=radorif(ibub)
      volperbub=rad(ibub)*rad(ibub)*rad(ibub)*pi*1.33333
      gmass(ibub)=volperbub*deng(ibub)
      endif
c
c This section defines the radius of the bubble, depending
c on the air content and the local pressure
c
      if (mconti(ibub).ne.0) then
      deng(ibub)=(pmean+pabs)/(runiv*(temp+tabs))
      vol=gmass(ibub)/deng(ibub)
      rad(ibub)=(0.75*vol/pi)**(0.3333)
      endif
c
      return
      end
c
c ***************************************************************
c
      subroutine shifter
      include'com-bubble'
c
c This subroutine properly shifts all bubble arrays by initnum
c place to the right, to make space for the new to-be-injected
c bubbles
c
      do 7793 ibib=ibubble+initnum,initnum+1,-1
      xcen(ibib)=xcen(ibib-initnum)
      ycen(ibib)=ycen(ibib-initnum)
      zcen(ibib)=zcen(ibib-initnum)
      xcenm1(ibib)=xcenm1(ibib-initnum)
      ycenm1(ibib)=ycenm1(ibib-initnum)
      zcenm1(ibib)=zcenm1(ibib-initnum)
      xcenm2(ibib)=xcenm2(ibib-initnum)
      ycenm2(ibib)=ycenm2(ibib-initnum)
      zcenm2(ibib)=zcenm2(ibib-initnum)
      rad(ibib)=rad(ibib-initnum)
      mconti(ibib)=mconti(ibib-initnum)
      id(ibib)=id(ibib-initnum)
      ilomem(ibib)=ilomem(ibib-initnum)
      memoryi(ibib)=memoryi(ibib-initnum)
      memoryj(ibib)=memoryj(ibib-initnum)
      memoryk(ibib)=memoryk(ibib-initnum)
      ububble(ibib)=ububble(ibib-initnum)
      vbubble(ibib)=vbubble(ibib-initnum)
      wbubble(ibib)=wbubble(ibib-initnum)
      ububbm1(ibib)=ububbm1(ibib-initnum)
      vbubbm1(ibib)=vbubbm1(ibib-initnum)
      wbubbm1(ibib)=wbubbm1(ibib-initnum)
      ububbm2(ibib)=ububbm2(ibib-initnum)
      vbubbm2(ibib)=vbubbm2(ibib-initnum)
```

```
      wbubbm2(ibib)=wbubbm2(ibib-initnum)
      deng(ibib)=deng(ibib-initnum)
      xmass(ibib)=xmass(ibib-initnum)
      gmass(ibib)=gmass(ibib-initnum)
      brtime(ibib)=brtime(ibib-initnum)
 7793 continue
      return
      end
c
c ***************************************************************
c
      subroutine swapthem
      include'com-bubble'
c
c This subroutine investigates the state of each bubble
c (existant or inexistant) and rearranges all bubble matrices to carry
c only existant bubbles.
c
      itrash=0
      j=0
      do 8331 i=1,ibubble
      if (id(i).ne.0) then
      j=j+1
      xcensw(j)=xcen(i)
      ycensw(j)=ycen(i)
      zcensw(j)=zcen(i)
      xcenm1sw(j)=xcenm1(i)
      ycenm1sw(j)=ycenm1(i)
      zcenm1sw(j)=zcenm1(i)
      xcenm2sw(j)=xcenm2(i)
      ycenm2sw(j)=ycenm2(i)
      zcenm2sw(j)=zcenm2(i)
      radsw(j)=rad(i)
      mcontisw(j)=mconti(i)
      idsw(j)=id(i)
      memoryisw(j)=memoryi(i)
      memoryjsw(j)=memoryj(i)
      memoryksw(j)=memoryk(i)
      ububblesw(j)=ububble(i)
      vbubblesw(j)=vbubble(i)
      wbubblesw(j)=wbubble(i)
      ububbm1sw(j)=ububbm1(i)
      vbubbm1sw(j)=vbubbm1(i)
      wbubbm1sw(j)=wbubbm1(i)
      ububbm2sw(j)=ububbm2(i)
      vbubbm2sw(j)=vbubbm2(i)
      wbubbm2sw(j)=wbubbm2(i)
      dengsw(j)=deng(i)
      xmasssw(j)=xmass(i)
      gmasssw(j)=gmass(i)
      brtimesw(j)=brtime(i)
      else
      itrash=itrash+1
      endif
 8331 continue
      if(j.ne.(ibubble-itrash)) write (*,*) 'Debug here'
      i=0
      do 8332 j=1,ibubble-itrash
      i=i+1
      xcen(i)=xcensw(j)
      ycen(i)=ycensw(j)
```

```
      zcen(i)=zcensw(j)
      xcenm1(i)=xcenm1sw(j)
      ycenm1(i)=ycenm1sw(j)
      zcenm1(i)=zcenm1sw(j)
      xcenm2(i)=xcenm2sw(j)
      ycenm2(i)=ycenm2sw(j)
      zcenm2(i)=zcenm2sw(j)
      rad(i)=radsw(j)
      mconti(i)=mcontisw(j)
      id(i)=idsw(j)
      memoryi(i)=memoryisw(j)
      memoryj(i)=memoryjsw(j)
      memoryk(i)=memoryksw(j)
      ububble(i)=ububblesw(j)
      vbubble(i)=vbubblesw(j)
      wbubble(i)=wbubblesw(j)
      ububbm1(i)=ububbm1sw(j)
      vbubbm1(i)=vbubbm1sw(j)
      wbubbm1(i)=wbubbm1sw(j)
      ububbm2(i)=ububbm2sw(j)
      vbubbm2(i)=vbubbm2sw(j)
      wbubbm2(i)=wbubbm2sw(j)
      deng(i)=dengsw(j)
      xmass(i)=xmasssw(j)
      gmass(i)=gmasssw(j)
      brtime(i)=brtimesw(j)
 8332 continue
c
c set the new number of active bubbles
c
      ibubble=ibubble-itrash
c
      return
      end
c
c ****************************************************************
c
      subroutine cdreyn(velrel)
      include'com-bubble'
      dimension re(15),cd(15)
c
c This subroutine computes the bubble Reynolds number and the
c corresponding drag coefficient.
c
c Original Cdrag graph
c
      re(1)=-0.001
      cd(1)=30000.
      re(2)=0.1
      cd(2)=300.
      re(3)=1.
      cd(3)=30.
      re(4)=10.
      cd(4)=4.9
      re(5)=100.
      cd(5)=1.1
      re(6)=1000.
      cd(6)=0.5
      re(7)=10000.
      cd(7)=0.41
      re(8)=100000.
```

```
      cd(8)=0.49
      re(9)=220000.
      cd(9)=0.4
      re(10)=400000.
      cd(10)=0.1
      re(11)=500000.
      cd(11)=0.08
      re(12)=600000.
      cd(12)=0.1
      re(13)=1000000.
      cd(13)=0.18
      re(14)=5000000.
      cd(14)=0.31
      re(15)=5000000000.
      cd(15)=0.31
c
c Bubble Reynolds number
c
      reynb=2.*rad(ibub)*velrel*denw/viscw
      do 1,i=1,14
      if ((reynb.le.re(i+1)).and.(reynb.gt.re(i))) then
      cdcoef=cd(i)+(reynb-re(i))*(cd(i+1)-cd(i))/(re(i+1)-re(i))
      goto 2
      endif
  1   continue
c     write(*,*) 'Danger in Cd interp.',
c    +umean,ububble(ibub),reynb,velrel,rad(ibub)

c
  2   continue
      return
      end
c
c ***************************************************************
c
      subroutine create_br(gnew,rnew,ibr,ibgive)
      include'com-bubble'
c
c This subroutine arranges the new buuble in its temporary arrays
c
c The following hardcoded variable representing where the new
c bubble center is located, needs refinement with some short of
c statistical modelling.
      radjump=1.5
      if(xcen(ibgive).lt.0.) then
      xcenbr(ibr)=xcen(ibgive)+rnew*radjump
      xcenm1br(ibr)=xcenm1(ibgive)+rnew*radjump
      xcenm2br(ibr)=xcenm2(ibgive)+rnew*radjump
      endif
      if(xcen(ibgive).ge.0.) then
      xcenbr(ibr)=xcen(ibgive)-rnew*radjump
      xcenm1br(ibr)=xcenm1(ibgive)-rnew*radjump
      xcenm2br(ibr)=xcenm2(ibgive)-rnew*radjump
      endif
      if(ycen(ibgive).lt.0.) then
      ycenbr(ibr)=ycen(ibgive)+rnew*radjump
      ycenm1br(ibr)=ycenm1(ibgive)+rnew*radjump
      ycenm2br(ibr)=ycenm2(ibgive)+rnew*radjump
      endif
      if(ycen(ibgive).ge.0.) then
      ycenbr(ibr)=ycen(ibgive)-rnew*radjump
```

```
      ycenm1br(ibr)=ycenm1(ibgive)-rnew*radjump
      ycenm2br(ibr)=ycenm2(ibgive)-rnew*radjump
      endif
      if(zcen(ibgive).lt.0.) then
      zcenbr(ibr)=zcen(ibgive)+rnew*radjump
      zcenm1br(ibr)=zcenm1(ibgive)+rnew*radjump
      zcenm2br(ibr)=zcenm2(ibgive)+rnew*radjump
      endif
      if(zcen(ibgive).ge.0.) then
      zcenbr(ibr)=zcen(ibgive)-rnew*radjump
      zcenm1br(ibr)=zcenm1(ibgive)-rnew*radjump
      zcenm2br(ibr)=zcenm2(ibgive)-rnew*radjump
      endif
c
      gmassbr(ibr)=gnew
      radbr(ibr)=rnew
      mcontibr(ibr)=mconti(ibgive)
      idbr(ibr)=id(ibgive)
      ilomemibr(ibr)=ilomem(ibgive)
      memoryibr(ibr)=memoryi(ibgive)
      memoryjbr(ibr)=memoryj(ibgive)
      memorykbr(ibr)=memoryk(ibgive)
      ububblebr(ibr)=ububble(ibgive)
      vbubblebr(ibr)=vbubble(ibgive)
      wbubblebr(ibr)=wbubble(ibgive)
      ububbm1br(ibr)=ububbm1(ibgive)
      vbubbm1br(ibr)=vbubbm1(ibgive)
      wbubbm1br(ibr)=wbubbm1(ibgive)
      ububbm2br(ibr)=ububbm2(ibgive)
      vbubbm2br(ibr)=vbubbm2(ibgive)
      wbubbm2br(ibr)=wbubbm2(ibgive)
      dengbr(ibr)=deng(ibgive)
      return
      end
c
c ***************************************************************
c
      subroutine newbubs(ibroken)
      include'com-bubble'
c
c This subroutine adds the bubbles created by breakup to the
c main bubble arrays
c
      do 44 i=ibubble+1,ibubble+ibroken
      gmass(i)=gmassbr(i-ibubble)
      xcen(i)=xcenbr(i-ibubble)
      ycen(i)=ycenbr(i-ibubble)
      zcen(i)=zcenbr(i-ibubble)
      xcenm1(i)=xcenm1br(i-ibubble)
      ycenm1(i)=ycenm1br(i-ibubble)
      zcenm1(i)=zcenm1br(i-ibubble)
      xcenm2(i)=xcenm2br(i-ibubble)
      ycenm2(i)=ycenm2br(i-ibubble)
      zcenm2(i)=zcenm2br(i-ibubble)
      rad(i)=radbr(i-ibubble)
      mconti(i)=mcontibr(i-ibubble)
      id(i)=idbr(i-ibubble)
      ilomem(i)=ilomemibr(i-ibubble)
      memoryi(i)=memoryibr(i-ibubble)
      memoryj(i)=memoryjbr(i-ibubble)
      memoryk(i)=memorykbr(i-ibubble)
```

```
      ububble(i)=ububblebr(i-ibubble)
      vbubble(i)=vbubblebr(i-ibubble)
      wbubble(i)=wbubblebr(i-ibubble)
      ububbm1(i)=ububbm1br(i-ibubble)
      vbubbm1(i)=vbubbm1br(i-ibubble)
      wbubbm1(i)=wbubbm1br(i-ibubble)
      ububbm2(i)=ububbm2br(i-ibubble)
      vbubbm2(i)=vbubbm2br(i-ibubble)
      wbubbm2(i)=wbubbm2br(i-ibubble)
      deng(i)=dengbr(i-ibubble)
      brtime(i)=100000000.
   44 continue
c
c set new bubble number
c
      ibubble=ibubble+ibroken
      return
      end
c
c ************************************************************
c
      subroutine complamda
      include'com-bubble'
c
c This subroutine computes the lamda of the flow,
c lamda=Qair/(Qair/Qwater)
c
      qair=0.
      do i=1,initnum
      qair=qair+cfsair(i)
      enddo
      qwater=(3.14159*scale*scale/4.)*velscale
      xlamda=qair/(qair+qwater)
      write(*,*) 'Qwater is ',qwater
      write(*,*) 'Qair is ',qair
      write(*,*) 'Lamda coefficient is', xlamda
      return
      end
c
c ************************************************************
c
      subroutine geom
      include'com-bubble'
c
c This subroutine pre-computes the directions of the
c grid cells, to speed up subsequent force computations
c
c
      do ilo=1,iblock
      do i=1,ix(ilo)-1
      do j=1,iy(ilo)-1
      do k=1,iz(ilo)-1
c
      xdir1(i,j,k,ilo)=
     +(x(i+1,j,k,ilo)-x(i,j,k,ilo))/sqrt(
     +(x(i+1,j,k,ilo)-x(i,j,k,ilo))**2+(y(i+1,j,k,ilo)-y(i,j,k,ilo))**2+
     +(z(i+1,j,k,ilo)-z(i,j,k,ilo))**2)
c
      ydir1(i,j,k,ilo)=
     +(y(i+1,j,k,ilo)-y(i,j,k,ilo))/sqrt(
     +(x(i+1,j,k,ilo)-x(i,j,k,ilo))**2+(y(i+1,j,k,ilo)-y(i,j,k,ilo))**2+
```

```
     +(z(i+1,j,k,ilo)-z(i,j,k,ilo))**2)
c
      zdir1(i,j,k,ilo)=
     +(z(i+1,j,k,ilo)-z(i,j,k,ilo))/sqrt(
     +(x(i+1,j,k,ilo)-x(i,j,k,ilo))**2+(y(i+1,j,k,ilo)-y(i,j,k,ilo))**2+
     +(z(i+1,j,k,ilo)-z(i,j,k,ilo))**2)
ccc
      xdir2(i,j,k,ilo)=
     +(x(i,j+1,k,ilo)-x(i,j,k,ilo))/sqrt(
     +(x(i,j+1,k,ilo)-x(i,j,k,ilo))**2+(y(i,j+1,k,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j+1,k,ilo)-z(i,j,k,ilo))**2)
c
      ydir2(i,j,k,ilo)=
     +(y(i,j+1,k,ilo)-y(i,j,k,ilo))/sqrt(
     +(x(i,j+1,k,ilo)-x(i,j,k,ilo))**2+(y(i,j+1,k,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j+1,k,ilo)-z(i,j,k,ilo))**2)
c
      zdir2(i,j,k,ilo)=
     +(z(i,j+1,k,ilo)-z(i,j,k,ilo))/sqrt(
     +(x(i,j+1,k,ilo)-x(i,j,k,ilo))**2+(y(i,j+1,k,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j+1,k,ilo)-z(i,j,k,ilo))**2)
ccc
      xdir3(i,j,k,ilo)=
     +(x(i,j,k+1,ilo)-x(i,j,k,ilo))/sqrt(
     +(x(i,j,k+1,ilo)-x(i,j,k,ilo))**2+(y(i,j,k+1,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j,k+1,ilo)-z(i,j,k,ilo))**2)
c
      ydir3(i,j,k,ilo)=
     +(y(i,j,k+1,ilo)-y(i,j,k,ilo))/sqrt(
     +(x(i,j,k+1,ilo)-x(i,j,k,ilo))**2+(y(i,j,k+1,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j,k+1,ilo)-z(i,j,k,ilo))**2)
c
      zdir3(i,j,k,ilo)=
     +(z(i,j,k+1,ilo)-z(i,j,k,ilo))/sqrt(
     +(x(i,j,k+1,ilo)-x(i,j,k,ilo))**2+(y(i,j,k+1,ilo)-y(i,j,k,ilo))**2+
     +(z(i,j,k+1,ilo)-z(i,j,k,ilo))**2)

      enddo
      enddo
      enddo
      enddo
      write(*,*) 'Completed computing grid line directions'
      return
      end
c
      subroutine dimensions
      include'com-bubble'
c
c This subroutines scales the CFD grid and solution
c from model to full scale
c Pressure must be scaled and dimensionalized (using sigmaP)
c BEFORE the grid is scaled
c
c
c scale pressure using sigma values
c
      call  pres_scl_dt
c
c
c scale velocities, grid, k & e
c
```

```
c ck constant of k-e turbulence model, needed for transformation
      ck=0.09
c
      do 34 ilo=1,iblock
      do 34 i=1,ix(ilo)
      do 34 j=1,iy(ilo)
      do 34 k=1,iz(ilo)
c
c If k-w turbulence model, transform omega to epsilon
      if(iturbul.eq.1) then
      eps(i,j,k,ilo)=eps(i,j,k,ilo)*ck*xk(i,j,k,ilo)
      endif
c
      u(i,j,k,ilo)=u(i,j,k,ilo)*velscale
      v(i,j,k,ilo)=v(i,j,k,ilo)*velscale
      w(i,j,k,ilo)=w(i,j,k,ilo)*velscale
      xk(i,j,k,ilo)=xk(i,j,k,ilo)*(velscale**2)
      eps(i,j,k,ilo)=eps(i,j,k,ilo)*(velscale**3)
      x(i,j,k,ilo)=x(i,j,k,ilo)*scale
      y(i,j,k,ilo)=y(i,j,k,ilo)*scale
      z(i,j,k,ilo)=z(i,j,k,ilo)*scale
c
 34   continue
c
      return
      end
c
c
      subroutine pres_scl_dt
c
c This soubroutine dimensionalizes the pressure in the
c draft tube and sets it to real full-scale values.
c Dimensionalization procedure provided by:
c Garry Franke and Justin Hall, Voith Hydro Inc.
c
      include'com-bubble'
c

      do 100 ilo=1,iblock
      do 100 i=1,ix(ilo)
      do 100 j=1,iy(ilo)
      do 100 k=1,iz(ilo)
c
      sigmaT=(-p(i,j,k,ilo)/(denw*gi*Href))-HLoverHREF+
     +VSQexover2gHref-VsqDTE
      pinterm=PrefProto*(sigmaP-sigmaT)+Pvapor +
     +denw*gi*(Zref-z(i,j,k,ilo))*scale
c
      p(i,j,k,ilo)=pinterm
c
  100 continue
      return
      end

      subroutine walls(itrajend,memi,memj,memk,membl)
      include'com-bubble'
c
c This subroutine determines if the fish has hit the wall and
c in that case, bounces the fish back in the flow
c
      ii=memi
```

```
      jj=memj
      kk=memk
      ll=membl
c
      ii1=ii-ibbac
      jj1=jj-ibbac
      kk1=kk-ibbac
      ii2=ii+ibbac
      jj2=jj+ibbac
      kk2=kk+ibbac
      do iii=ii1,ii2
      iiii=iii
      do jjj=jj1,jj2
      jjjj=jjj
      do kkk=kk1,kk2
      kkkk=kkk

      if(iiii.gt.ix(ll)) iiii=ix(ll)
      if(iiii.lt.1) iiii=1
      if(jjjj.gt.iy(ll)) jjjj=iy(ll)
      if(jjjj.lt.1) jjjj=1
      if(kkkk.gt.iz(ll)) kkkk=iz(ll)
      if(kkkk.lt.1) kkkk=1

      if (iwall(iiii,jjjj,kkkk,ll).eq.0) then
      itrajend=1
      goto 767
      endif

      enddo
      enddo
      enddo
      return
c
c
 767  continue
c



      return
      end
c
c
      subroutine sources

c This soubroutine computes momentum source terms from the drag
c bubble put on the flow, to be used with a RANS solver
c
      include'com-bubble'
c
c
c Initialize
c
      do 21 ilo=1,iblock
      do 21 i=1,ix(ilo)
      do 21 j=1,iy(ilo)
      do 21 k=1,iz(ilo)
      xsource(i,k,k,ilo)=0.
      ysource(i,k,k,ilo)=0.
      zsource(i,k,k,ilo)=0.
      xsourcesm(i,k,k,ilo)=0.
```

```
      ysourcesm(i,k,k,ilo)=0.
      zsourcesm(i,k,k,ilo)=0.
   21 continue


c Computing sources
c
      write(*,*) 'Computing sources'
      do 9999 ibub=1,ibubble
c
c prepape data for locate
      xbub(1,1)=xcen(ibub)
      ybub(1,1)=ycen(ibub)
      zbub(1,1)=zcen(ibub)
      iconti=mconti(ibub)
c
c find position of bubble
      call locate(iconti)
c compute ambient flow field
      call ambient
c compute forces exerted on bubble
c bubble frontal area
      volfac=pi*rad(ibub)*rad(ibub)
c viscous forces
      if(mconti(ibub).eq.1) then
      uxrel=ububble(ibub)-umean
      vyrel=vbubble(ibub)-vmean
      wzrel=wbubble(ibub)-wmean
      velrel=sqrt(uxrel**2+vyrel**2+wzrel**2)
      call cdreyn(velrel)
      dragforce=0.5*cdcoef*volfac*denw*velrel**2
      xvisc=-(dragforce/velrel)*uxrel
      yvisc=-(dragforce/velrel)*vyrel
      zvisc=-(dragforce/velrel)*wzrel
      endif
c
c Place viscous forces in appropriate cell
c
      xsource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))=
     +xsource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))+
     +xvisc
      ysource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))=
     +ysource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))+
     +yvisc
      zsource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))=
     +zsource(memoryi(ibub),memoryj(ibub),memoryk(ibub),ilomem(ibub))+
     +zvisc

 9999 continue
c
c Apply smoothing
c
      write(*,*) 'Applying smoothing'
c
      do ismo=1,nosmooth
      write(*,*) 'Pass ', ismo

      do 22 ilo=1,iblock
c
      do 22 i=2,ix(ilo)-1
```

```
      do 22 j=2,iy(ilo)-1
      do 22 k=2,iz(ilo)-1
c
      x1=xsource(i,j,k,ilo)
      x2=(1./6.)*(xsource(i+1,j,k,ilo)+xsource(i,j+1,k,ilo)+
     +xsource(i,j,k+1,ilo)+xsource(i-1,j,k,ilo)+
     +xsource(i,j-1,k,ilo)+xsource(i,j,k-1,ilo))
      x3=(1.-smoothfact)*x1 + smoothfact*x2
      xsourcesm(i,j,k,ilo)=x3
c
      y1=ysource(i,j,k,ilo)
      y2=(1./6.)*(ysource(i+1,j,k,ilo)+ysource(i,j+1,k,ilo)+
     +ysource(i,j,k+1,ilo)+ysource(i-1,j,k,ilo)+
     +ysource(i,j-1,k,ilo)+ysource(i,j,k-1,ilo))
      y3=(1.-smoothfact)*y1 + smoothfact*y2
      ysourcesm(i,j,k,ilo)=y3
c
      z1=zsource(i,j,k,ilo)
      z2=(1./6.)*(zsource(i+1,j,k,ilo)+zsource(i,j+1,k,ilo)+
     +zsource(i,j,k+1,ilo)+zsource(i-1,j,k,ilo)+
     +zsource(i,j-1,k,ilo)+zsource(i,j,k-1,ilo))
      z3=(1.-smoothfact)*z1 + smoothfact*z2
      zsourcesm(i,j,k,ilo)=z3
c
 22   continue
c
c  Boundary conditions
c
      do 18 ilo=1,iblock
c
      do i=2,ix(ilo)-1
      do j=2,iy(ilo)-1
      xsourcesm(i,j,1,ilo)=xsourcesm(i,j,2,ilo)
      xsourcesm(i,j,iz(ilo),ilo)=xsourcesm(i,j,iz(ilo)-1,ilo)
      ysourcesm(i,j,1,ilo)=ysourcesm(i,j,2,ilo)
      ysourcesm(i,j,iz(ilo),ilo)=ysourcesm(i,j,iz(ilo)-1,ilo)
      zsourcesm(i,j,1,ilo)=zsourcesm(i,j,2,ilo)
      zsourcesm(i,j,iz(ilo),ilo)=zsourcesm(i,j,iz(ilo)-1,ilo)
      enddo
      enddo
c
      do i=2,ix(ilo)-1
      do k=1,iz(ilo)
      xsourcesm(i,1,k,ilo)=xsourcesm(i,2,k,ilo)
      xsourcesm(i,iy(ilo),k,ilo)=xsourcesm(i,iy(ilo)-1,k,ilo)
      ysourcesm(i,1,k,ilo)=ysourcesm(i,2,k,ilo)
      ysourcesm(i,iy(ilo),k,ilo)=ysourcesm(i,iy(ilo)-1,k,ilo)
      zsourcesm(i,1,k,ilo)=zsourcesm(i,2,k,ilo)
      zsourcesm(i,iy(ilo),k,ilo)=zsourcesm(i,iy(ilo)-1,k,ilo)
      enddo
      enddo
c
      do j=1,iy(ilo)
      do k=1,iz(ilo)
      xsourcesm(1,j,k,ilo)=xsourcesm(2,j,k,ilo)
      xsourcesm(ix(ilo),j,k,ilo)=xsourcesm(ix(ilo)-1,j,k,ilo)
      ysourcesm(1,j,k,ilo)=ysourcesm(2,j,k,ilo)
      ysourcesm(ix(ilo),j,k,ilo)=ysourcesm(ix(ilo)-1,j,k,ilo)
      zsourcesm(1,j,k,ilo)=zsourcesm(2,j,k,ilo)
      zsourcesm(ix(ilo),j,k,ilo)=zsourcesm(ix(ilo)-1,j,k,ilo)
      enddo
```

```
      enddo
c
 18   continue
c
c
      enddo


      open(1,file='SOURCES.dat')
      write(1,*) 'TITLE = "SOURCES"'
      write(1,*) 'VARIABLES= X,Y,Z,SX,SY,SZ,Sxsm,SYsm,SZsm'
      do ilo=1,iblock
      write(1,*)
     +'ZONE T="1",I=',ix(ilo),',J=',iy(ilo),',K=',iz(ilo),' F=block'
      write(1,*) (((x(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((y(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*) (((z(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((xsource(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((ysource(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((zsource(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((xsourcesm(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((ysourcesm(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      write(1,*)
     +(((zsourcesm(i,j,k,ilo),i=1,ix(ilo)),j=1,iy(ilo)),k=1,iz(ilo))
      enddo
      close(1)
c
c
      open(2,file='SOURCES-RES')
c
      do 23 ilo=1,iblock
c
      do 23 i=1,ix(ilo)
      do 23 j=1,iy(ilo)
      do 23 k=1,iz(ilo)
c
      write(2,500) ilo,i,j,k,xsource(i,j,k,ilo),ysource(i,j,k,ilo),
     +zsource(i,j,k,ilo),xsourcesm(i,j,k,ilo),ysourcesm(i,j,k,ilo),
     +zsourcesm(i,j,k,ilo)
c
  500 format(4i5,6e12.4)
  23  continue
c
      close(2)
c
c
      return
      end
```

The program `multi-id-bubbles.f`

A listing of a typical post-processing program is presented here. This particular code, `multi-id-bubbles.f`, operates on the output file `RESULTS-***.plt` and creates a TECPLOT data file with one individual zone for each bubble. Each bubble is represented as a sphere of radius proportional to that of the corresponding bubble. The bubble `id(*)` is maintained is the output file, information that is useful when tracking the fate of bubbles coming from specific injection ports.

```
      dimension xfish(36,36),yfish(36,36),zfish(36,36)
      pi=3.14159
      ixf=11
      jxf=11
      scale=.8
      open(88,file='MULTI-ID.dat')
      write(88,*) 'TITLE = "BUBBLES"'
      write(88,*) 'VARIABLES= X, Y, Z,id'
      open(1,file='RESULTS-00300.plt')
      write(*,*) 'no of bubbles?'
      read(*,*) ibubble
      do 1 n=1,ibubble
      read(1,*)time,isteper,id,x,y,z,r,g
c     if(y.gt.18.) goto 1
      write(*,*) x,y,z,r,id
  19  format(f11.5,i8,i3,4f11.4,e12.4)


      theta=0.
      dth=pi/float(ixf-1)
      dphi=2.*pi/float(jxf-1)
      do i=1,ixf
      phi=0.
      do j=1,jxf
           xfish(i,j) =   cos(theta)
           yfish(i,j) = sin(theta)*sin(phi)
           zfish(i,j) = sin(theta)*cos(phi)
      xfish(i,j)=r*scale*xfish(i,j)+x
      yfish(i,j)=r*scale*yfish(i,j)+y
      zfish(i,j)=r*scale*zfish(i,j)+z
      phi=phi+dphi
        enddo
      theta=theta+dth
        enddo

      if(id.lt.6) iid=1
      if((id.ge.6).and.(id.le.10)) iid=2
      if((id.le.11).and.(id.le.15)) iid=2
      if(id.gt.16) iid=4
      write(88,*) 'ZONE T="bubble", I=',ixf,', J=',jxf,' F=block'
      write(88,*) ((xfish(ii,jj),ii=1,ixf),jj=1,jxf)
      write(88,*) ((yfish(ii,jj),ii=1,ixf),jj=1,jxf)
      write(88,*) ((zfish(ii,jj),ii=1,ixf),jj=1,jxf)
      write(88,*) ((iid,ii=1,ixf),jj=1,jxf)
  1   continue
      close(88)
      close(1)
      end
```